LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
Il6r
no.445-450

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

FEB 1 9 1976

FEB 1 9 RECD

L161—O-1096

Digitized by the Internet Archive
in 2013

http://archive.org/details/procedurefordete449maru

St. 4
IlGN
no. 449

Report No. 449

Math

A PROCEDURE FOR DETECTING INTERSECTIONS AND ITS APPLICATION

by

Kiyoshi Maruyama

May, 1971

THE LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 449

A PROCEDURE FOR DETECTING INTERSECTIONS AND ITS APPLICATION

by

Kiyoshi Maruyama

May, 1971

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the Department of Computer Science at the University of Illinois at Urbana-Champaign, Urbana, Illinois; the National Science Foundation under NSF Grant GJ-328.

ACKNOWLEDGEMENT

The author wishes to thank Dr. J. Nievergelt, Professor of Computer Science at the University of Illinois, for suggesting the problem and his helpful discussion and comments. Thanks are extended to Miss Sue Cook who helped in getting this report finished.

TABLE OF CONTENTS

		Page
1.	INTRODUCTION	1
2.	INTERSECTION DETECTION PROCEDURES FOR GEOMETRIC OBJECTS	2
	2.1 P-C FUNCTION APPROACH	2
	2.2 LP AND IP APPROACH	5
	2.3 GEOMETRIC APPROACH	6
	2.4 SPACE PARTITION APPROACH	6
3.	GEOMETRIC PROCEDURE FOR DETECTING INTERSECTIONS	11
	3.1 PRELIMINARY DISCUSSION	11
	3.2 FACE-TO-FACE INTERSECTION DETECTION	15
	3.3 FURTHER DISCUSSION	20
	3.4 ALGORITHM AND EXPERIMENTAL RESULT	25
4.	A PROBLEM IN PATH FINDING	29
	4.1 STATE SEARCH AND STATE CONTIGUITY	30
	4.2 PATH SEARCHING	34
	4.3 SEARCH TREE PRUNING	37
	4.4 EVALUATION FUNCTION AS A DISTANCE EVALUATOR	37
	4.5 EXAMPLES OF DISTANCE EVALUATION FUNCTION	40
	4.6 SIMULATION RESULTS OF GOAL SEEKING AUTOMATON	49
5.	CONCLUSION	71
LIST OF REFERENCES		72

1. INTRODUCTION

A problem of both practical and theoretical interest is to determine if an object can be moved from one position α to another position β, inside a complicated geometric structure. In some problems there will be no path from α to β and some others there will be several paths of which perhaps the shortest must be determined. A restricted problem of this type, the two dimensional sofa problem has been studied by Howden [17].

To solve such a problem it is basically important to have a very efficient algorithm for determining whether two bodies intersect, since a moving object cannot share the same space with some other moving objects or with obstacles at the same time. This intersection detection problem between physical objects has been studied by Comba[1], for convex objects with differentiable surfaces.

The plan of this report is as follows. In chapter 2 we introduce several intersection detection procedures for geometric objects, and an intersection detection procedure by means of face-to-face intersection analysis will be discussed in Chapter 3. As an application of the intersection detection procedure developed in Chapter 3, we will disucss a path finding problem in geometrically constrained space in Chapter 4.

2. INTERSECTION DETECTION PROCEDURES FOR GEOMETRIC OBJECTS

In this chapter, we describe several procedures to detect intersections between objects. The representation of objects and its data structure [8 , 19] are heavily dependent upon the class of objects as well as a detection procedure used.

2.1 P-C FUNCTION APPROACH

Let us first describe the procedure which has been developed by Comba [1] for detecting intersections of convex regions in 3-space by means of a pseudo-characteristic (P-C) function.

Given n objects, each of which is bounded by one or more surfaces, the problem of deciding whether all the objects have a common intersection may be formulated as the problem of finding whether a system of inequalities (in general nonlinear) has a feasible solution. In the approach followed here, the expressions that represent the bounding surfaces of all the objects are combined into a single pseudocharacteristic function, $G(x,y,z)$, which has the property that the region where $G \leq 0$, if such a region exists, is a close approximation to the region where all the objects intersect. The problem is thus reduced to that of finding whether there are any points in space where G takes on nonpositive values.

A). An object is the intersection of $k \geq 1$ regions R_i where each R_i is defined by an inequality $g_i(x,y,z) \leq 0$, and the function g_i has the following properties:

(i) It is convex

(ii) It is differentiable

(iii) It is normalized so that $|\text{grad } g_i| \geq 1$ on the surface $g_i = 0$, and

(iv) Given a bounded region of space, grad g_i is bounded there.

The p-c function G of the intersection of n objects is defined in terms of the g_i functions for the objects by

$$v_i = (g_i^2 + t^2)^{1/2} + g_i \qquad (1)$$

$$V = \sum v_i \qquad (2)$$

$$G = \frac{1}{2}\left(V - \frac{t^2}{V}\right) + c \qquad (3)$$

where t anc c are small positive numbers. The boundedness of the objects is not a serious restriction from a practical viewpoint. The convexity assumption is the most restrictive, since it implies that only convex objects can be defined. Comba proved that G is a convex function of the g_i and hence of the coordinate variables. Convexity and boundedness imply further that, G has exactly one minimum. It is possible to compute lower bounds for the values of G without actually finding its minimum. The assumption of differentiability makes it possible to use a gradient method in the search for negative values of G and the computation of lower bounds to G.

B). A sequence of points approaching the minimum of G is constructed by means of a descent procedure. The search is terminated as soon as

(i) a negative value of G is obtained, indicating that there is an intersection; or

(ii) the minimum of G is obtained and if it is positive, there is no intersection; if bounds on the size of the objects are known, it is possible to compute lower bounds for G, and the search can be stopped if a positive lower bound is established.

The central idea of the method is that of applying the transformation (1) to each of the function g_i that define the various regions R_i. The function V_i is always positive and it is an increasing function of g_i. If $t \leq 1$, V_i also has the property that

$$V_i = \begin{cases} 2g_i + O(t^2) & g_i > 0 \\ t & g_i = 0 \\ O(t^2) & g_i < 0 \,. \end{cases}$$

Consider the expression

$$G^* = \frac{1}{2}\left(V - \frac{t^2}{V}\right)$$

we can easily verify that the transformation from V to G^* is the inverse of the transformation from g_i to V_i. It can be also seen that

$$G^* \begin{cases} < 0 & V < t \\ = 0 & V = t \\ > 0 & V > t. \end{cases}$$

Hence, except for the vicinity of the surfaces $g_i = 0$, G* will be negative inside the intersection of the regions $g_i < 0$ and positive outside that intersection. The reader may see [1] for further discussion.

If all of the functions g_i are linear, the intersection detection problem reduces to a linear programming problem, i.e. deciding whether a set of linear inequalities has a feasible solution. In this case one would expect a priori that a linear programming algorithm will solve the problem more efficiently than an algorithm based on the p-c function.

Comba assumed convex objects with differentiable surfaces but we hereafter assume a more restricted class of objects namely, polygons for 2-space and polyhedra for 3-space. This restriction is reasonable because many objects

can be represented as the union of (convex or nonconvex) polyhedra in 3-space and detecting intersection of such unions in an immediate extention of the case for a single pair of polyhedra.

2.2 LP AND IP APPROACHES

When one assumes polyhedral objects to be convex or to be unions of convex polyhedra, a set of linear inequalities represents a convex object. To detect an intersection between two objects, this algorithm (i.e. the phase 1 of Simplex Method) searches for the existence of feasible solutions to the joint set of linear inequalities of two convex polyhedra. Therefore, if one object Ω consists of the union of N convex polyhedra and the other object Ω' consists of N' convex polyhedra, then it is necessary to solve (N x N') joint sets of inequalities to conclude whether or not Ω and Ω' intersect. Furthermore, if there are M such objects in space to be solved for intersection, it is necessary to determine M(M-1)/2 object-object intersection detections.

However, allowing new integer variables (i.e. 0-1) we get one composite system by Integer Programming formulation.

Let Ω_i, i = 1,2, ...M, be M objects and let for each Ω_i, $g_i \leq 0$ denotes the corresponding set of inequalities. Since the placement of M objects is possible only if

$$\Omega_i \cap \Omega_j = \emptyset \quad \text{for i and j, } i \neq j.$$

we have the following Integer Programming problem formulation with the sacrifice of introducing new 0-1 integer variables, δ_{ij} for $i \neq j$.

$$G_{ij} \leq \delta_{ij} U_{ij} \quad \text{for } i \neq j$$

$$\sum_i \sum_j \delta_{ij} = \frac{M(M-1)}{2} - 1$$

$$\delta_{ij} = 0 \text{ or } 1$$

where G_{ij} denotes a joint set of inequalities of Ω_i and Ω_j, U_{ij} is an upper bound of G_{ij}. Thus, if the above composite system have a feasible solution then such placement of M objects is impossible, otherwise the placement is possible.

Unfortunately, even though we get one composite system to solve, if M get large then the formulation to Integer Programming problem become somewhat cumbersome as well as it increase the number of new variables which should be introduced greatly.

2.3 GEOMETRIC APPROACH

This algorithm employs a face-to-face intersection analysis, which first eliminates those faces which cannot possibly intersect the other polyhedra, and then makes a pairwise analysis of the remaining faces. This method will be discussed in the following chapter.

2.4 SPACE PARTITION APPROACH

This procedure partitions the bounded space into smaller and smaller cubes, and decides at each stage whether or not there can possibly be an intersection of polyhedra in the cube currently being determined. Many large cubes may usually be disposed of quickly. Those which cannot are partitioned further, if necessary down to the desired limit of resolution. A similar approach has been used for the hidden line problem [41].

There are three cut planes we consider,

a). X-cut: a partition of a cube by a plane which is parallel to the Y-Z plane,

b). Y-cut: a partition of a cube by a plane which is parallel to the Z-X plane, and

c). Z-cut: a partition of a cube by a plane which is parallel to the X-Y plane.

The decision procedure of which cut should be chosen for the next partitioning simply depends upon the last two cuts which have been used for the partitioning to get the cube, e.g., if last two cuts are an X-cut and a Y-cut then a Z-cut may be chosen for the next partition. Of course one may choose the next cut plane by considering the characteristics of the cube being cut.

At each stage of the partition, information about points, edges, and faces in the two partitioned cubes is kept for use in further partitioning as well as in detecting intersection. The algorithm stops partitioning a cube as soon as the size of the partitioned cubes become smaller than or equal to a predefined cube size, called resolution cube. It then checks whether or not such a cube contains portions of more than one object. If it does then the algorithm concludes that those objects have a "pseudo-intersection." The cube will be disposed from the stack of cubes, SOC, which is shown in Fig. 1.

The algorithm stops for further partition of a particular cube being determined currently as soon as it detects that the cube contains no object or only a portion of one object. Such a cube will also be disposed from the SOC, where the SOC contains only those cubes which require further partitioning to detect intersections.

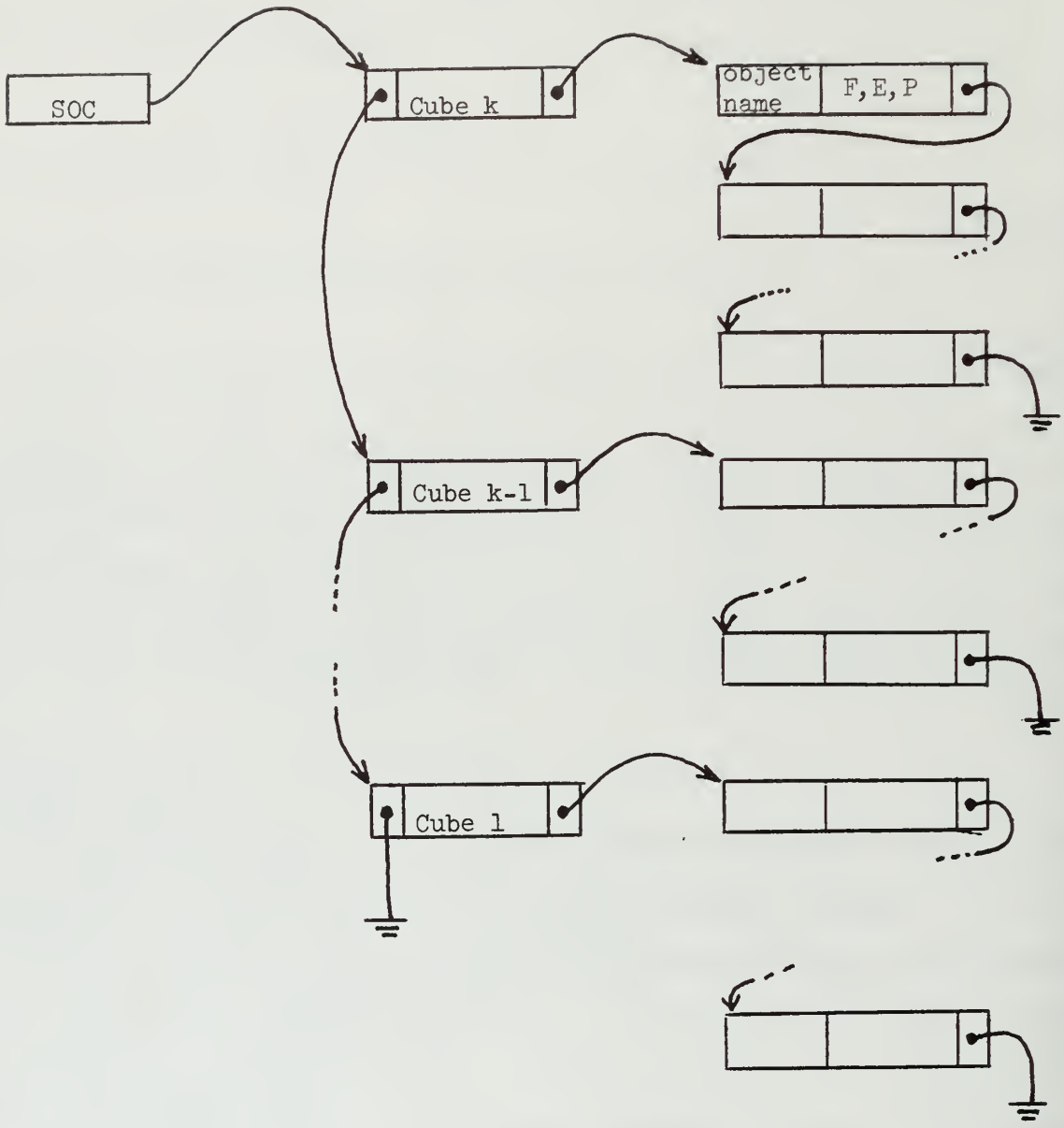


Fig. 1 The stack of cubes, SOC.

The algorithm terminates completely either when the SOC becomes empty or when M(M-1)/2 results are found, where M is the number of objects in the space.

It is quite often happens that some cubes in the SOC may contain portions of some objects such that those objects have been determined to intersect in an earlier stage of partitioning of some other cubes which were in the SOC. Thus the algorithm disposes of those cubes in the SOC which only contain objects whose intersections have been determined.

Fig. 1 illustrates a situation of the SOC at some stage.

To get an idea of the space partition approach, an example of the 2-space case (3-space case is a relatively simple extension of 2-space), where objects are denoted by polygons, is shown in Fig. 2. Cube 1 contains objects Ω_1, Ω_3, Ω_4, and Ω_6 while Cube 2 contains objects Ω_1, Ω_2, and Ω_5 after a X-cut partition has been performed. Cube 2 is further partitioned into Cube 2-1 and Cube 2-2 by Y-cut, and the former contains Ω_1 and Ω_2 while the latter contains only Ω_5, thus Cube 2-2 is disposed from the SOC. At this stage there are two cubes, Cube 2-1 and Cube 1 in the SOC.

We have the following two choices to select for the next cube partitioning from the stack: the cube at the top of the SOC or the cube which has the largest volume in the SOC. For example, for the first case, Cube 2-1 may be partitioned further to Cube 2-1-1 and Cube 2-1-2 at the third stage, while Cube 1 may be partitioned further to Cube 1-1 and Cube 1-2 for the second case as illustrated in Fig. 2.

It is interesting to notice that the first selection method corresponds to a tree search of depth first (see Chapter 4) in endorder or preorder, and the second selection method corresponds to a tree search of topdown in parallel.

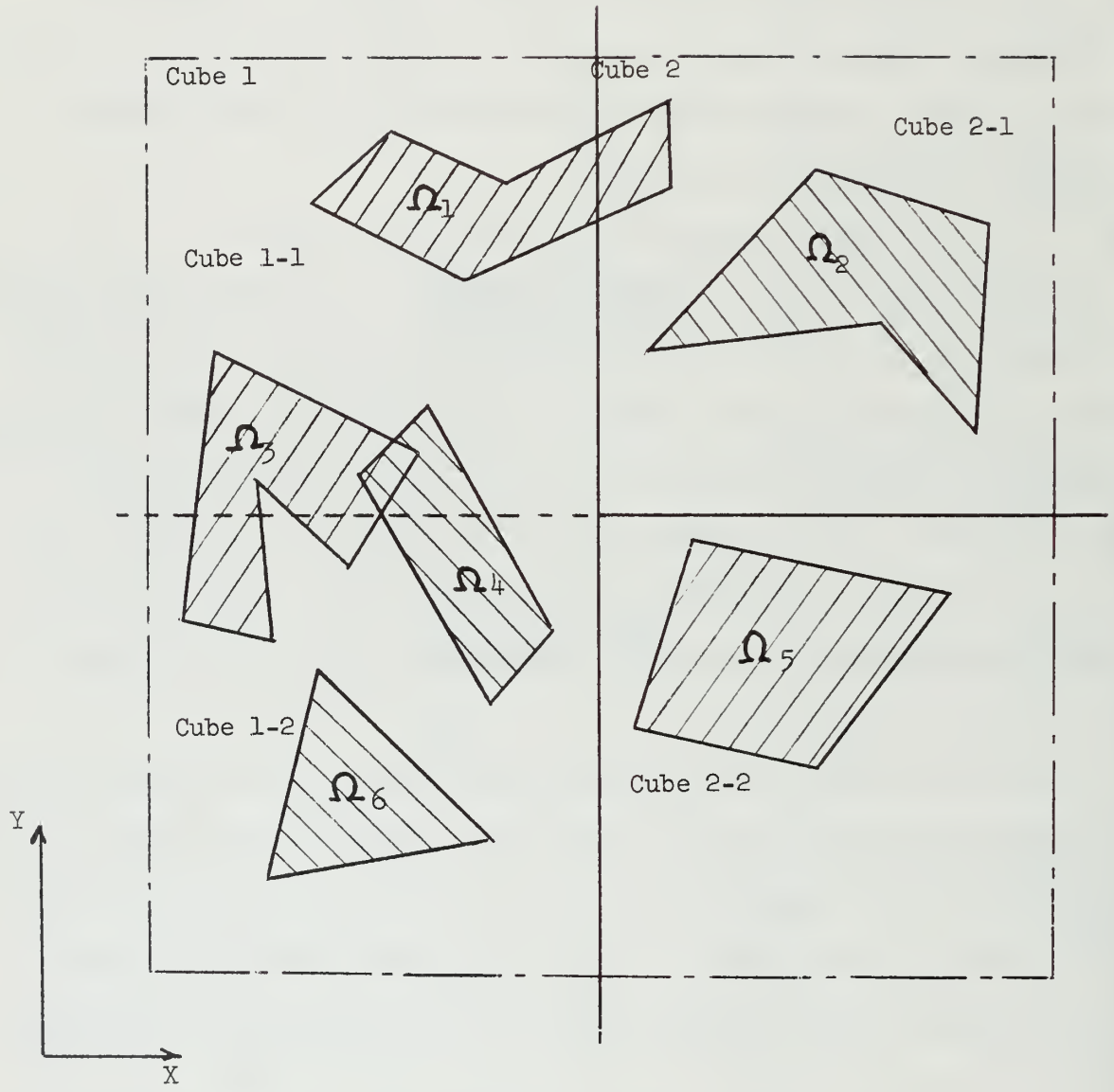


Fig. 2 An example of the space partition approach for 2-space case with polygonal objects.

3. GEOMETRIC PROCEDURE FOR DETECTING INTERSECTIONS

In this chapter, we discuss a direct approach, called face-to-face intersection detection procedure, which first eliminates those faces of two objects which cannot possibly intersect each other, and then makes a pairwise analysis on the remaining faces.

We divide objects which are polyhedra or unions of polyhedra into two classes: objects and obstacles. Objects can move around in the 3-space, while obstacles cannot and create a restricted, i.e., constrained, space for objects.

3.1 PRELIMINARY DISCUSSION

We use a four-tuple $\Omega(P, E, F; \bar{p})$ to denote an object (or an obstacle) which is a polyhedron in 3-space, where P is a set of points, E is a set of edges, F is a set of faces and $\bar{p}$ is a point called c-point defined as the average of the points of the object Ω:

$$\bar{p} = (\sum_{i=1}^{n} p_i)/n \qquad \text{where } P = \{p_1, p_2, \ldots, p_n\}.$$

A three-tuple $\Omega(P, E; \bar{p})$ may be used to denote a polygonal object in 2-space. Hereafter we consider polyhedral objects because similar properties of the following properties for polyhedral objects are applied to polygonal objects.

A minimal sphere, denoted by $\hat{S}(R)$ of an object Ω is defined as the smallest sphere of center $\bar{p}$ which contains Ω. Its radius is denoted by R. $\check{S}(r)$ denotes the maximal sphere of center $\bar{p}$ which is contained in Ω. Its radius is denoted by r. In general, $\check{S}(r)$ may be undefined for a non-convex object Ω since $\bar{p}$ may not be contained in Ω.

$$\hat{S}: \qquad R = \max_{p_i \in P} \{d(p_i, \bar{p})\}$$

$$\check{S}: \qquad r = \min_{f_j \in F} \{d(f_j, \bar{p})\}$$

where d denotes the Euclidian distance between two points or between a point and a plane.

Let $\Omega(P, E, F; \bar{p})$ and $\Omega'(P', E', F'; \bar{p}')$ be two objects whose minimal and maximal spheres are $\hat{S}(R)$, $\check{S}(r)$ and $\hat{S}(R')$, $\check{S}(r')$, respectively. Then Ω and Ω' are said to be strongly separated if $D > R + R'$, and are said to intersect strongly if $D < r + r'$, where $D = d(\bar{p}, \bar{p}')$.

The minimal box $\Pi(I_x, I_y, I_z)$ of an object Ω is the smallest rectangular parallelepiped whose three pairs of faces are parallel to X-Y plane, Y-Z plane, and Z-X plane, respectively, such that it contains the object Ω, where

$$I_x = [\inf_x(\Omega), \sup_x(\Omega)]$$

$$I_y = [\inf_y(\Omega), \sup_y(\Omega)] \quad \text{and}$$

$$I_z = [\inf_z(\Omega), \sup_z(\Omega)].$$

In practical cases we use $\Pi(I_x^*, I_y^*, I_z^*)$ instead of $\Pi(I_x, I_y, I_z)$ for the minimal box of Ω, where

$$I_x \subset I_x^* \qquad I_x^* - I_x = 2t$$

$$I_y \subset I_y^* \qquad I_y^* - I_y = 2t$$

$$I_x \subset I_z^* \qquad I_z^* - I_z = 2t$$

and $t > 0$ is called a tolerance.

Property 1.

Let Ω and Ω' be two polyhedral objects whose minimal boxes are $\Pi(I_x, I_y, I_z)$ and $\Pi'(I_x', I_y', I_z')$, respectively. Then a necessary condition for $\Omega \cap \Omega' \neq \Phi$ is $\Pi \cap \Pi' \neq \Phi$.

Let us further assume $\Omega \not\subset \Omega'$, $\Omega' \not\subset \Omega$, then the property 1 leads us to the following.

Property 2.

$\Omega \cap \Omega' \neq \Phi$ if and only if there exist at least a pair of faces (f_i, f_j'), $f_i \in F$, $f_j' \in F'$ such that $f_i \cap f_j' \neq \Phi$, where $f_i \cap (\Pi \cap \Pi') \neq \Phi$ and $f_j' \cap (\Pi \cap \Pi') \neq \Phi$.

Property 2 implies the outline of our geometric approach to detect intersection between two polyhedral objects.

It is enough to examine these <u>pairs of faces</u> (f_i, f_j'), where $f_i \in F$, $f_j' \in F$ and f_i, f_j' are at least partially contained in the intersection of two minimal boxes Π and Π', called the <u>solution box</u> and denoted by $\dot{\Pi}$, to determine whether or not Ω and Ω' intersect. We may expect that the introduction of the solution box reduces greatly the number of pairs of faces which should be considered further for face-to-face intersection analysis, and it makes our analytic approach competetive to the other approaches discussed in the preceding chapter. The solution box $\dot{\Pi}$ of two minimal boxes Π and Π' is defined only if the following conditions hold.

$$\max(\inf_x(\Omega), \inf_x(\Omega')) < \min(\sup_x(\Omega), \sup_x(\Omega'))$$

$$\max(\inf_y(\Omega), \inf_y(\Omega')) < \min(\sup_y(\Omega), \sup_y(\Omega'))$$

and

$$\max(\inf_z(\Omega), \inf_z(\Omega')) < \min(\sup_z(\Omega), \sup_z(\Omega')).$$

Now we encounter to the following two questions.

(a) How to extract those faces of two objects which intersect the solution box?

(b) How to determine whether or not a pair of faces intersect?

For the first question, one may use an approach to find "strictly only" those faces of two objects which intersect the solution box. But this leads us to the detection of intersection between the solution box and each face of objects. Even though this kind of intersection detection procedure may be implemented simpler than the one between two given objects, it still has to contain some cumbersome process. Therefore, instead of finding strictly only those faces which have intersection with the solution box, we extract those which may "possibly" intersect with the solution box. We consider the following simple approach to solve the question, where the solution box of the given two objects is denoted by

$\dot{\Pi}(\dot{I}_x,\dot{I}_y,\dot{I}_z) \neq \Phi$. We also assume that each face of objects is bounded by its minimal box, called a face-minimal box (c.f object-minimal box). Thus our extraction approach becomes to extract those faces such that the intersection of face-minimal box and $\dot{\Pi}$ is non-empty.

Some other properties to reduce the number of faces which should be determined for face-to-face intersection detection are discussed in section 3.3. The second question of determining the intersection of a pair of faces is described in the following section.

3.2 FACE-TO-FACE INTERSECTION DETECTION

Let us think about a polygonal face f in 3-space, where the plane of f, denoted by g_f, is the hyperplane which contains the face f. Then it is clear to see that any straight line L on g_f intersects an even number of edges of f (see Fig. 3).

Let f_i and f_j be two non-parallel polygonal faces whose hyperplanes are g_{f_i} and g_{f_j}, respectively. Let us also assume that L_0 denotes the intersection line of these two hyperplanes (see Fig. 6). By arranging intersections between L_0 and edges of f_i and f_j in geometrical order from the left to the right, we form a sequence of two different types of intersections; one for those on f_i and the other on f_j. Then f_i and f_j are said to have <u>even parity</u> if these two types of points are juxtaposed multiples of two by two each other. In other words, if 0 and 1 correspond to intersection points between L_0 and edges of f_i and L_0 and edges of f_j, respectively, then such a 0-1 sequence has an even parity if the sequence is contained in $(00 + 11)^*$, otherwise it has <u>non-even parity</u>.

In Fig. 4, examples of an even and a non-even parities on L_0 are shown. The state diagram illustrated in Fig. 5 shows how to recognize those sequences of even parity. Those sequences which end up at state S_1 are even parity and those encountered in state S_4 are non-even parity.

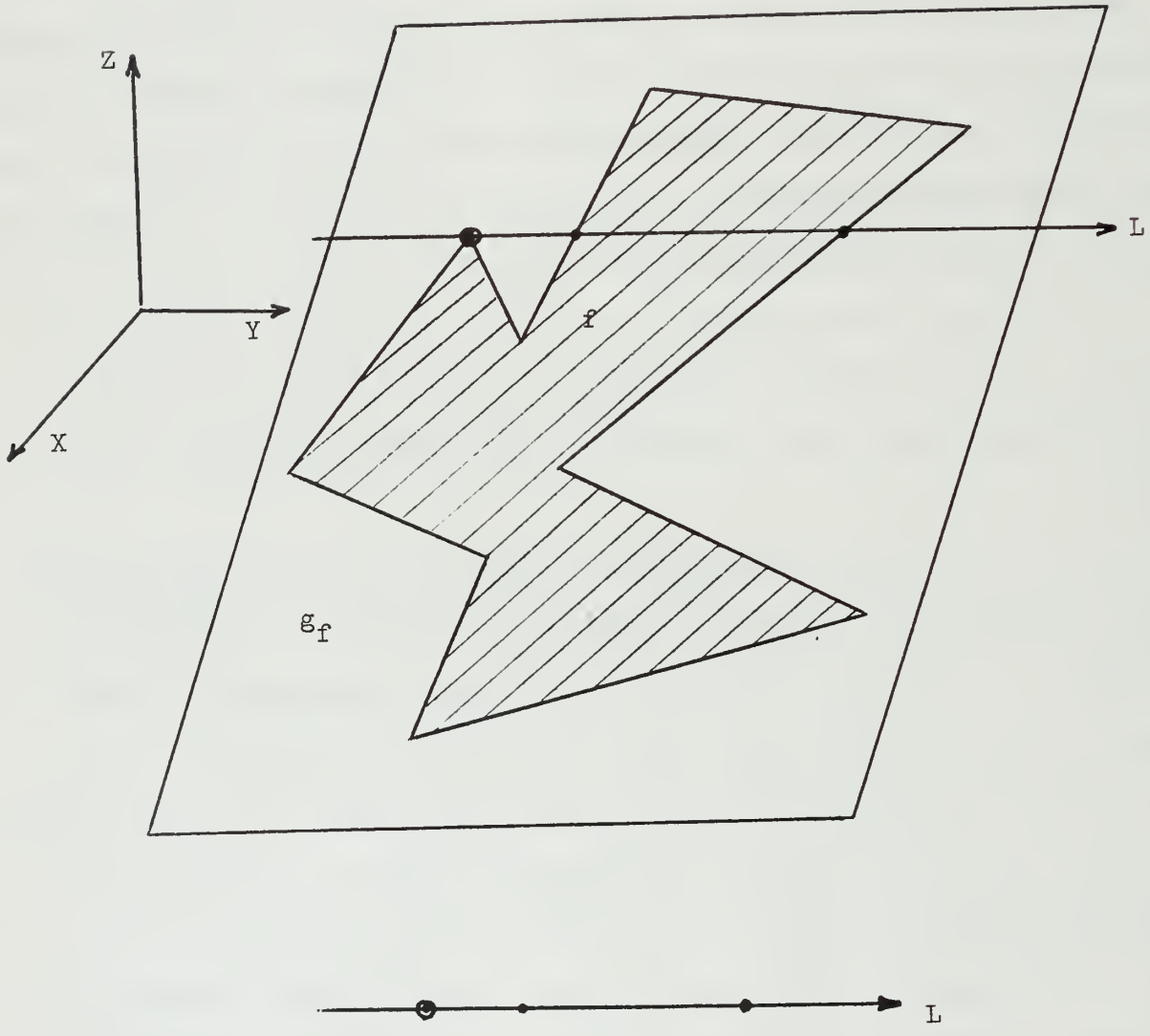
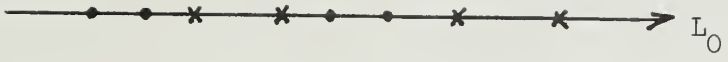
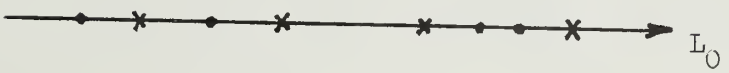


Fig. 3 The plane g_f of a face f.



(a). Even parity.



(b). Non-even parity.

Fig. 4 Even and non-even parities on L_0.

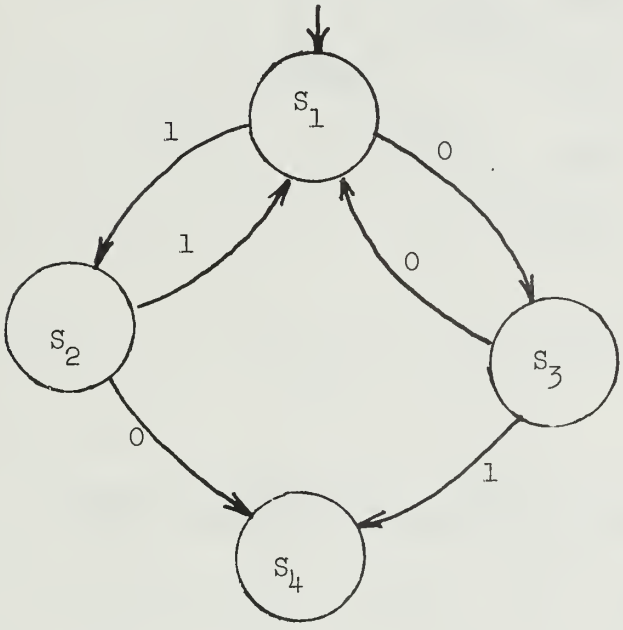


Fig. 5 The state diagram to recognize even parities, i.e., sequences of the form $(00 + 11)^*$.

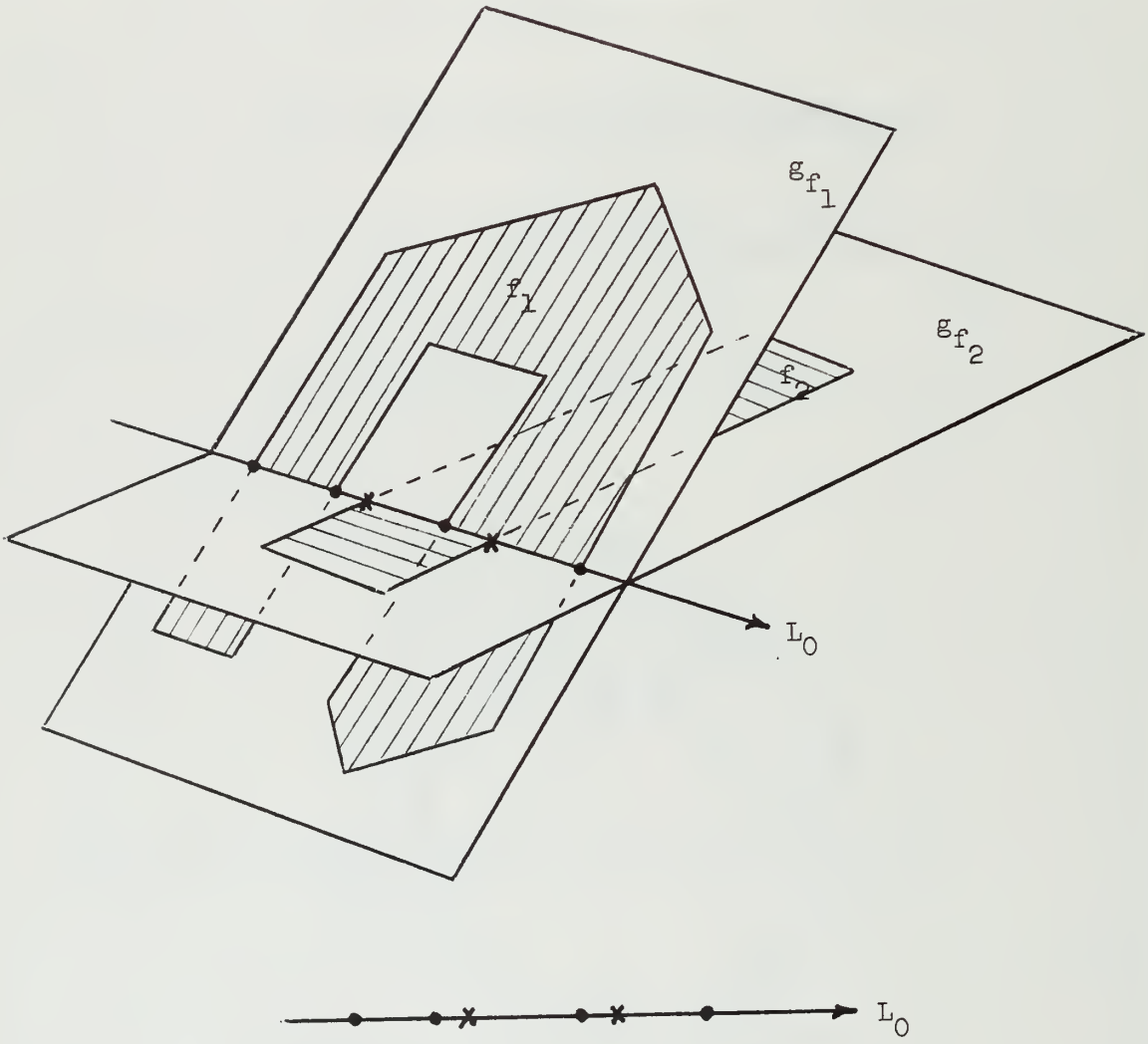


Fig. 6 A face-to-face intersection: nOn-even parity.

Theorem 1.

Let f_i and f_j be two non-parallel faces. Then f_i and f_j have no intersectio if and only if their parity is even, i.e., they intersect if and only if the parity is non-even.

Proof:

(separation $\rightarrow$ even parity).

Let us assume that there exist a sequence S whose parity is non-even and f_i and f_j separate.

$$S = s_1 s_2 \cdots s_i \cdots s_n \qquad n\text{:even number}$$

$s_i = 0$ or 1 for all i, $s_i = 0$ for f_i and

$s_i = 1$ for f_j,

$$\sum_{s_i=0} = \text{even}, \qquad \sum_{s_i=1} = \text{even}.$$

Since S is non-even parity, then there exist at least one subsequence S' in S such that $s_j = 0$ for $s_j \epsilon S'$ and $|S'|$ is odd. Let us denote such a subsequence of maximum cardinarity k, where k is odd, by

$$S' = s_{i_1} s_{i_2} s_{i_3} \cdots\cdots\cdots s_{i_k} = \underbrace{000 \cdots 0}_{k\ 0\text{'s}}$$

Then there exist $s_j = 0$ in S-S' such that either (s_j, s_{i_1}) or (s_{i_k}, s_j) forms an interval in L_0 which is contained in f_i. Without loss of generality, let (s_j, s_{i_1}) form such an interval, then it is clear to see that there exist a subsequence S" of all 1's between s_j and s_{i_1}. This says f_i and f_j have intersection in the interval corresponds to (s_j, s_{i_1}), which leads us to a contradiction.

(even parity $\longrightarrow$ separation)

This is clear from the definition of even parity since intervals formed by two by two selection of neighboring cross points on L_0 for f_i and those intervals for f_j have no intersection. (Q.E.D.)

Theorem 1 shows how to determine intersection between two faces by means of the recognition of even parity.

To generate the sequence for a pair of faces f_i and f_j, we first evaluate the intersection line L_0 of g_{f_i} and g_{f_j}. Then choose a sufficiently large interval which may contain all intersections on L_0 with f_i and f_j. From one end of the interval toward the other end, slide a small piece of segment, called a bullet, on L_0 and detect intersection between the bullet and edges of f_i and f_j. When such an intersection is determined then generate 0 or 1 depending on the bullet intersects an edge of f_i or an edge of f_j. At each generation of either 0 or 1, the parity state changes between states which are illustrated in Fig. 5. The bullet may be slided until the other end of the interval if necessary.

3.3 FURTHER DISCUSSION

We introduce some other concepts which may be used to make a set of these pairs extracted by means of solution box, as small as possible. In other words we try to make the number of face pairs, which should be determined for face-to-face intersection to conclude intersection of two given objects, small, and consequently we save the computation time for intersection detection.

Let $\Omega(P,E,F)$ and $\Omega'(P',E',F')$ be two polyhedral objects. If there exists a face $f \in F \cup F'$ such that the plane g_f of f separates Ω and Ω' into two different half-spaces, then it is clear to see the two objects have no intersection. This existence of such a face is determined by the following way. Without loss of generality, let us assume $f \in F'$, let $\dot{\Pi}$ denote the solution box of Ω and Ω' and let P_s and P'_s be $P_s \subset \dot{\Pi}$ and $P'_s \subset \dot{\Pi}$, respectively, where $P_s \subseteq P$ and $P'_s \subseteq P'$. Then if one

of the following conditions hold,

(a) $g_f(p_i) > 0$ for $p_i \epsilon P_s$, and
$g_f(p'_j) \leq 0$ for $p'_j \epsilon P'_s$

(b) $g_f(p_i) < 0$ for $p_i \epsilon P_s$, and
$g_f(p'_j) \geq 0$ for $p'_j \epsilon P'_s$

then Ω and Ω' have no intersection.

Two polygonal faces f_i and f_j are said to be mutually divisible if and only if the plane g_{f_i} of f_i divides points of f_j into two half-spaces and the plane g_{f_j} of f_j divides points of f_i into two half-spaces. A collection of such pairs is denoted by:

$$S_D = \{(f_i, f_j) \mid f_i \epsilon F,\ f_j \epsilon F',\ \text{and } f_i,\ f_j \text{ are mutually divisible}\}.$$

Then it is easy to see that if f_i and f_j are not mutually divisible, then f_i and f_j have no intersection. In other words, the mutual divisibility of two faces is a necessary condition that two faces intersect. Therefore, if S_D for given two object is empty then they have no intersection.

Let f be a face of a convex polyhedron $\Omega(P,E,F;\overline{p})$ and p be any point which is not contained in Ω. Then f is said to be visible at p if and only if $\text{sgn}(g_f(\overline{p})) = -\text{sgn}(g_f(p))$, where g_f is the plane of f.

Let $\Omega(P,E,F;\overline{p})$ and $\Omega'(P',E',F';\overline{p}')$ be two convex polyhedra. Then a pair of faces (f_i, f'_j), $f_i \epsilon F$, $f'_j \epsilon F'$, is said to be mutually visible if there exist points p_i, p'_j, p_i is in f_i and p'_j is in f'_j, such that f_i is visible at p'_j and f'_j is visible at p_i. Such a collection of pairs is denoted by:

$$S_r = \{(f_i, f'_j) \mid f_i \epsilon F,\ f'_j \epsilon F',\ f_i, f'_j \text{ are mutually visible}\}.$$

From the above definition of mutual visibility, it is easy to see that the mutual visibility of two faces f_i and f'_j is a necessary condition that f_i and f'_j intersect.

Theorem 2.

Let S_V denote a collection of pairs of faces which are mutually visible for two given convex polyhedra Ω and Ω'. Then $\Omega \subseteq \Omega'$ or $\Omega \supseteq \Omega'$ if and only if $S_V = \Phi$.

Proof: Let us assume $\Omega \supseteq \Omega'$ then there exist no point in any face of Ω' that some faces of Ω are visible because by the definition of visibility. Thus $S_V = \Phi$.

From the definition of mutual visibility, $S_V = \Phi$ implies either $p_i \subset \Omega'$ for all $p_i \in P$ or $p_j' \subset \Omega$ for all $p_j' \in P'$. Since Ω and Ω' are convex, we get either $\Omega \subseteq \Omega'$ or $\Omega \supseteq \Omega'$. Q.E.D.

Theorem 3.

Let Ω and Ω' be two polyhedra such that $\Omega \cap \Omega' \neq \Phi$, $\Omega \not\subseteq \Omega'$ and $\Omega \not\supseteq \Omega'$. Then there exist at least three pairs of faces in S_C whose parities are non-even, where

$$S_C = \begin{cases} S_B \cap S_D \cap S_V & \text{if } \Omega, \Omega' \text{ are convex} \\ S_B \cap S_D & \text{otherwise,} \end{cases}$$

S_B : pairs extracted by solution box

S_D : mutually divisible pairs

S_V : mutually visible pairs.

Proof: It is easy to see that $S_C \neq \Phi$ for the given condition. Let us assume neithe point of Ω nor Ω' contained in the other, then Ω has at least 2 faces to intersect Ω' and Ω' has at least 2 faces to intersect Ω. Thus there are at least 4 pairs of faces which are non-even parities.

Let us assume Ω contains a point p'_j of Ω' then since p'_j is at least 3-points (which means at p'_j at least three inequalities corresponding to faces become zero), there are at least three pairs of faces whose parities are non-even. (Q.E.D.)

The Vein diagram of mutually divisible pairs, mutually visible pairs, and pairs by solution box is illustrated in Fig. 7, where we assumed $\Omega \cap \Omega' \neq \Phi$, $\Omega \not\subseteq \Omega'$ and $\Omega \not\supseteq \Omega'$.

In some cases one may want to know to what degree two given objects intersect. Therefore, let us introduce some measures of the degree of penetration. To measure the degree of penetration of two polyhedral objects Ω and Ω', we define P-deg as

$$\text{P-deg}(\Omega,\Omega') = \max({}^{v}/_{V} , {}^{v}/_{V'})$$

where V, V' and v correspond to the volume of Ω, Ω' and $\Omega \cap \Omega'$, respectively. Therefore, if objects have no intersection then P-deg$(\Omega,\Omega') = 0$. If one object contains the other completely then it becomes 1, otherwise $0 < \text{P-deg}(\Omega,\Omega') < 1$.

Let us assume that space is bounded and digitized into an n x n x n point matrix, then the volume of an object may correspond to the cardinality of the point set. Using this cardinality, we define another measure of the degree of penetration, denoted by P-deg_n,

$$\text{P-deg}_n(\Omega,\Omega') = \max \left(\frac{|\Omega \cap \Omega'|}{|\Omega|} , \frac{|\Omega \cap \Omega'|}{|\Omega'|} \right)$$

and $\lim_{n\to\infty} \text{P-deg}_n(\Omega,\Omega') = \text{P-deg}(\Omega,\Omega')$,

where $|\ |$ indicates cardinality of an object which is digitized.

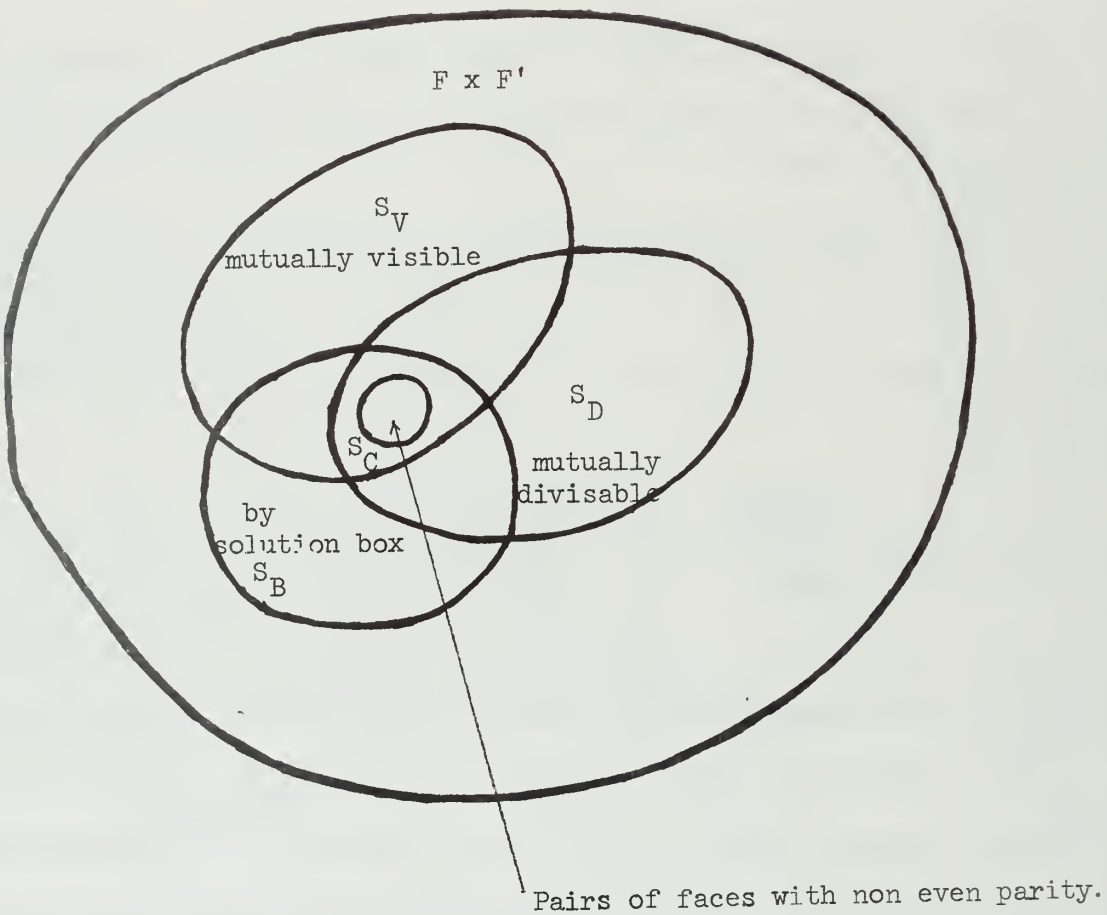


Fig. 7 Vein diagram of filtering processes on F x F'.

3.4 ALGORITHM AND EXPERIMENTAL RESULT

From the properties discussed in the preceding sections, we get the following algorithm PD which detects intersection between two polyhedral objects. A minor modification of the algorithm leads to the intersection detection of many objects as well as the measurement of the degrees of penetration.

The main scheme of this algorithm is to first find those pairs of faces, i.e., the candidate pairs, which possibly intersect by using the solution box (if the number of such pairs is quite large then the further partition of the solution box is carried out to minimize the number). Second to eliminate these pairs which obviously do not intersect from the candidate pairs by means of mutual divisibility and mutual visibility, and then apply face-to-face intersection detection on the remaining pairs.

Algorithm PD.

Step 1. Evaluate $\hat{S}(R)$, $\check{S}(r)$ and $\hat{S}(R')$, $\check{S}(r')$ for $\Omega(P,E,F;\overline{p})$ and $\Omega'(P',E',F';\overline{p}')$, respectively. Set $D = d(p, \overline{p}')$. If $D \geq R + R'$ then Ω and Ω' are strongly separated, terminate. If $D \leq r + r'$ then Ω and Ω' are strongly intersected, terminate.

Step 2. Evaluate the minimal boxes $\Pi(I_x,I_y,I_z)$ and $\Pi'(I_x,I_y,I_z)$ for Ω and Ω' respectively. Set $\dot{\Pi} = \Pi \cap \Pi'$. If $\dot{\Pi} = \Phi$ then Ω and Ω' have no intersection, terminate.

Step 3. Evaluate the following sets. F_s is a subset of F such that each face of F_s possibly intersects $\dot{\Pi}$, F'_s is a subset of F such that each face of F'_s possibly intersect $\dot{\Pi}$. P_s and P'_s are subsets of basic point sets P and P', respectively, such that each point in the set is in $\dot{\Pi}$.

If there exists a face f in $F_s \cup F'_s$ such that for all $p_i \epsilon P_s$, all $p_j \epsilon P'_s$ either one of the following pair of conditions holds, then terminate.

(a) $g_f(p_i) > 0$ and $g_f(p_i) \leq 0$,

(b) $g_f(p_i) < 0$ and $g_f(p_i) \geq 0$

where g_f is the plane of f.

Step 4. If the addition of two cardinalities, $|F_s| + |F'_s|$, is larger than M (where M is an integer about 10), then the following partition of the solution box $\dot{\Pi}$ is carried. Partition $\dot{\Pi}$ into $\dot{\Pi}_1$ and $\dot{\Pi}_2$ by the similar way as used in "Space Partition Approach" described in Chapter 2. Evaluate F_{s_1} and F_{s_2}, $F_{s_1} \cap F_{s_2}$ may not necessarily disjoint, such that each face of F_{s_1} possibly intersect $\dot{\Pi}_1$ and each face of F_{s_2} possibly intersect $\dot{\Pi}_2$. Similarly, evaluate F'_{s_1} and F'_{s_2} such that each face of F'_{s_1} possibly intersects $\dot{\Pi}_1$ and each face of F'_{s_2} possibly intersects $\dot{\Pi}_2$, respectively. If the above partition of $\dot{\Pi}$ carried then set $S_c = F_{s_1} \times F'_{s_1} \cup F_{s_2} \times F'_{s_2}$. Otherwise $S_c = F_s \times F'_s$.

Step 5 If $S_c = \Phi$, then Ω and Ω' have no intersection, terminate.

Step 6. Choose a pair (f_i, f'_j) from S_c, and set $S_c = S_c - (f_i, f_j)$. If the pair is not mutually divisible then the pair has no intersection, thus go back to Step 5. (the mutual visibility of the pair is determined if Ω and Ω' are convex polyhedra).

Step 7. Determine the parity of the pair, if it is non-even then the pair intersec thus terminate. Go back to Step 5.

It is easy to understand that if objects and obstacles are restricted to a class of convex polyhedra or disjoint unions of convex polyhedra then some other properties for convex polyhedra which have not been discussed in this paper may be used to minimize the cardinality of the set of candidate pairs which should be

determined for their parities. If objects and obstacles are restricted to a class of polyhedra with only convex faces, then the face-to-face intersection detection procedure, especially parity determination procedure, is greately simplified.

From the computation time aspect, we found that it is important to minimize the candidate pairs as much as possible since the time required for parity checks is greater than those for the other filtering processes.

In Fig. 8, we show an example of one object and four obstacles in 3-space which has been used as a data for Algorithm PD. The object called "UFO" is a non-convex polyhedron consists of 14 faces, 24 edges and 24 points. The obstacles called "HOUSE" and "TREE" are represented by disjoint unions of convex polyhedra. The HOUSE consists of 36 faces, 72 edges and 48 points. The TREE consists of 19 faces, 36 edges and 23 points. The obstacle called "ROCK" is a convex polyhedron consists of 9 faces, 16 edges and 9 points.

In this example, the object UFO was moved and rotated around the obstacles and detected intersection. The average run time for detecting intersection for the illustrated data example was about a quarter to a half second, where the algorithm was written in PL/1 and implemented on 360/75.

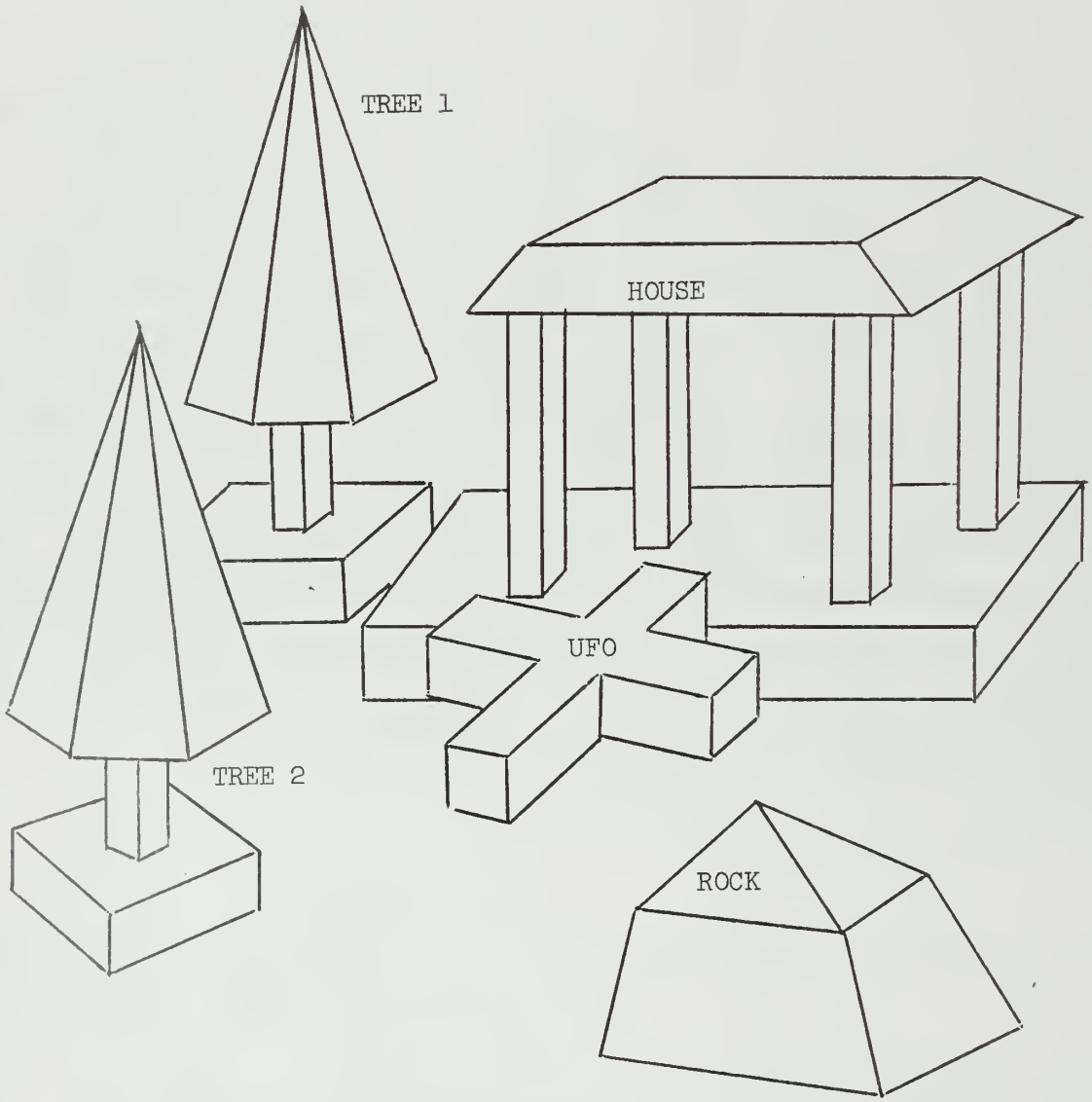


Fig. 8 An example of the simulated objects for 3-space.

4. A PROBLEM IN PATH FINDING

One of the most basically important studies in artificial intelligence is to find an efficient algorithm for the shortest path in a given state space. This area has been studied for many years and various shortest path algorithms and their applications may be seen in [3,4,5,9,10,15,17,20,21,23,27,28,30,31,34].

A problem of both practical and theoretical interest is to find efficient algorithms to determine if an object Ω_0 can be moved from one position p_s to another position p_t, inside a complicated geometric structure C. In some problems there will be no path from p_s to p_t and some others there will be several paths of which perhaps the shortest must be determined. A restricted problem of this type, the two-dimensional sofa problem [13, 24, 36], has been studied by Howden [17].

We are interested in a more generalized formulation of this problem as stated below. There is a given set S of states, which may not necessarily be specified explicitely, and a set $T = \{t_1, t_2, \ldots\ldots, t_k\}$ of transformation provided for Ω_0 on S. A transformation will be applicable to some, but not necessarily all, of the states. Thus the problem is to determine a sequence Γ of transformations,

$$\Gamma = t_{i_1} \cdot t_{i_2} \cdot t_{i_3} \cdot \cdots\cdots t_{i_n} ,$$

such that an application of Γ on Ω_0 moves Ω_0 from a state α to another state β, the target or the goal, and at each state Ω_0 is under certain given constraints.

This problem mainly consists of the following two parts. One is the "state-search" problem where we wish to find a state with some particular property and the other is the "path-search" problem where we seek a sequence of transformations between two states.

We assume an automaton A_0 of Ω_0 with no visual information but with whole information of territory as well as the shape of the A_0. It was pointed out by Ernst [9] that an automonous object must have a picture of the world, and a notion of its own location and state, in order to predict the consequence of its actions and hence to generate alternative courses of action, and decide among them. An internal representation of the environment has two fundamental characteristics that determine the amount of relevant dimensions of the environment and the resolution along any single dimension. The former although dictated largely by the nature of the environment itself, may also be a function of the given task.

We hereafter consider an automonous automaton A_0 of Ω_0 with transformations translations and rotation, and obstacles Ω_i, $i = 1,2,\ldots,m$ which form a constrained environment C for Ω_0, where each Ω_i for $i = 1,2,\ldots,m$ is restricted to an undeformable polygon. The assigned task on the automaton A_0 is to find a path from a given starting state α to another given goal state β which may be optimal under the satisfaction of the given environmental constraints.

4.1 STATE SEARCH AND STATE CONTIGUITY

When an automaton A_0 of Ω_0 moves from the present state α_i to the next state α_j, it must know whether Ω_0 can occupy α_j under environmental constraints, and should also determine whether or not an application of some transformations provided for A_0 achieve the motion from α_i to α_j, in some sense continuously. The first question corresponds to the problem of solving intersection between Ω_0 and obstacles Ω_i, $i = 1,2,\ldots,m$, in C. The second corresponds to the problem of solving the question that A_0 of Ω_0 can actually move from α_i to α_j under the satisfaction of environmental constraints, e.g., if some rotational transformations on Ω_0 are required to achieve α_j such rotations should be contiguous. The contiguity relation between two states will be defined later in this section.

A procedure for the detection of intersection between Ω_0 and Ω_i, $i = 1,2,\ldots,m$, which are polygonal obstacles, is a simplification of the intersection detection procedure developed in Chapter 3. Let Π_0 and Π_i, $i = 1,2,\ldots,m$, are the minimal boxes for Ω_0 and Ω_i, $i = 1,2,\ldots,m$, respectively. If $\Pi_0 \cap \Pi_i = \Phi$ for all i then Ω_0 has no intersection with any of obstacles. If $\Pi_0 \cap \Pi_i \neq \Phi$ for some i, then extract those edges of Ω_0 and Ω_i which may intersect with the solution box $\dot{\Pi}_{0_i} = \Pi_0 \cap \Pi_i$. If the number of edges which are extracted by the above process is quite large, then $\dot{\Pi}_{0_i}$ is further partitioned and the procedure minimizes the number of pairs of edges which should be determined for intersection.

Let (e_j, e_k) is a pair of edges such that $e_j \cap \dot{\Pi}_{0_i} \neq \Phi$ and $e_k \cap \dot{\Pi}_{0_i} \neq \Phi$, where $e_j \epsilon E_0$ of Ω_0 and $e_k \epsilon E_i$ of Ω_i. Then the mutual divisibility between e_j and e_k is determined to detect intersection between Ω_0 and Ω_i.

To discuss state contiguity, let us first consider an example which is illustrated in Fig. 9, where the automaton A_0 can occupy the next state α_3 from α_2 with a translation and rotation of degree $\pi/2$. But it is easy to see that the rotation cannot be achieved either at α_2 nor at α_3, i.e., even though α_3 satisfies the given constraints, transformation on Ω_0 from α_2 to α_3 is not achieved. Therefore, one way to achieve α_3, Ω_0 should be rotated by an angle $\pi/2$ at the previous state α_1 before A_0 applies translation on Ω_0 to α_3.

Two states $\alpha_1(x_1,y_1,\theta_1)$ and $\alpha_2(x_2,y_2,\theta_2)$ are said to be <u>contiguous</u> if A_0 has a translation $t_i : (x_1,y_1) \rightarrow (x_2,y_2)$ with $t_i^{-1} : (x_2,y_2) \rightarrow (x_1,y_1)$ and $\theta_1 \cap \theta_2 \neq \Phi$, where θ_1 and θ_2 denote admissible range of rotations at (x_1,y_1) and (x_2,y_2), respectively. Let $\alpha_3(x_3,y_3,\theta_3)$ be another state and assume α_2 and α_3 are contiguous. It is easy to see that, in general, the transitivity relation of contiguity does not hold, i.e., α_1 and α_2 contiguous and α_2 and α_3 contiguous does not imply α_1 and α_3 are contiguous through α_2. Fig. 10 shows the admissible range

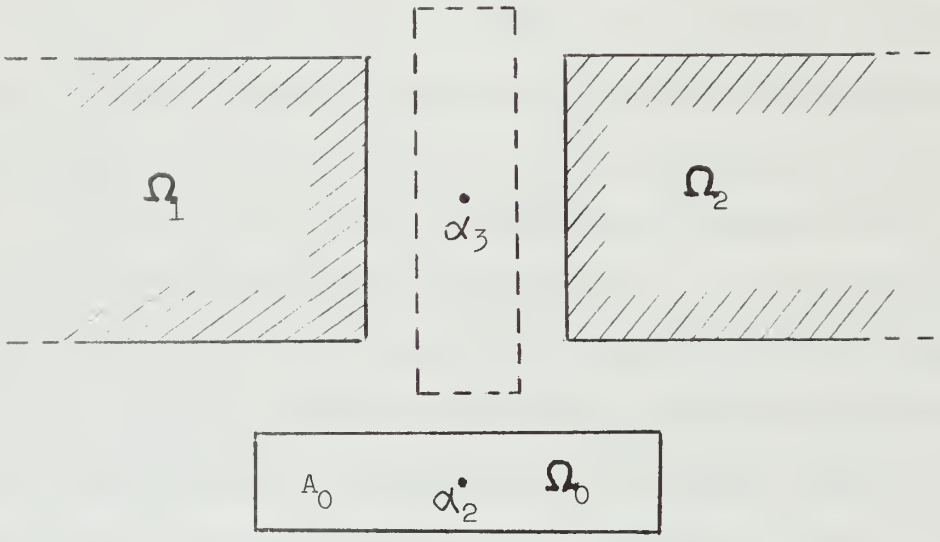


Fig. 9 State contiguity.

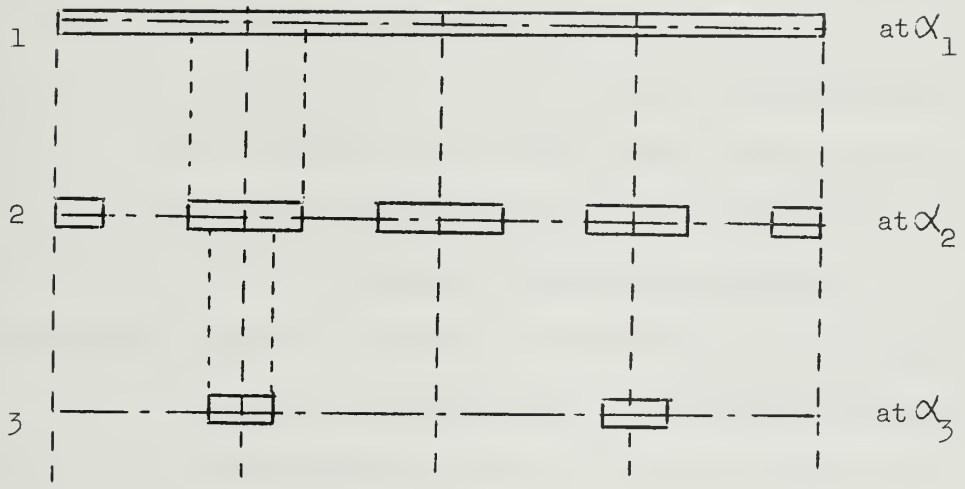


Fig. 10 Contiguity on rotational transformations.

of rotations at α_1, α_2, and α_3 which correspond to the illustration of Fig. 9. If θ_2 consists of a single interval of admissible rotation, then α_1, α_2 contiguous and α_2, α_3 contiguous imply the contiguity between α_1 and α_3 through α_2, i.e., the extistance of transformation from α_1 to α_3 through α_2 under the satisfaction of environmental constraints. This contiguity relation should be determined for each state if a transformation from one state to another require rotations.

4.2 PATH SEARCHING

We will consider heuristic tree search approaches while a straight forward exhaustive search procedure has been taken by Howden [17].

A heuristic tree search [3, 30, 31] consists of the following two parts: one is constructing a path tree for admissible states given by state search, and second, an evaluation function which estimates a future path distance (or the number of transformations which may be required to move automaton A_0 of Ω_0 from the state being considered) to the goal state.

Let us first consider the problem of searching a tree to find a given goal state. If it is necessary to search the whole tree, i.e., to find the node which is optimal in some sense (by optimal we mean the sequence Γ of transformations with minimal cardinality), the sequence in which the search is carried out is unimportant. If, however, we are content to discover just one state with the given property the sequence in which states are searched may be of great interest. We will consider the following tree searching.

Search Controlled by an Evaluation Function

In this method of search we may think of a "frontier" of those nodes whose successors have not been examined. This frontier is extended by choosing any node in it and examining its successors. The node chosen may be that which has the

minimum value of some functions called "evaluation functions" normally chosen on heuristic grounds. (Such a function may be stated as the maximum desirability and sometimes called "reluctance function".)

This method of search has been extensively discussed by Doran and Michie [3] who used it in their program called the "Graph Traverser". They have shown that given a suitable evaluation function it is an effective method of trackling a number of problems.

The main point to notice is that the next node whose successors are examined is not necessarily one of those produced at the previous move, i.e. the frontier will be pushed forward for a while in one region, but if the evaluation function indicates that further advances here are not as profitable as had been anticipated, some other part of the frontier may be extended.

Measures of Performance [35]

The three most interesting measures of tree search programs performance over a particular search of a graph are

(i) the length of the path produced ($|\Gamma|$)

(ii) the total number of nodes developed (D)

(iii) the total number of nodes added to the search tree (N).

Since every node on the constructed path except the last must have been developed, it follows that $|\Gamma| \leq N$. Denote the minimal path length for a given start and goal by $|\Gamma^*|$. This will also be the minimum number of developments which must be made to find a path from the start to the goal. Then we define the path efficiency as $|\Gamma^*|/|\Gamma|$. Similarly, we define the development efficiency as $|\Gamma^*|/D$. The calculation of both these efficiency requires a knowledge of $|\Gamma^*|$. Typically, however, $|\Gamma^*|$ will be unknown. The ratio $|\Gamma|/D$, which is the fraction of the total number of nodes developed which are incorporated into the actual path found, can always be calculated, and is of considerable importance. It will be

(a). low penetration.

(b). high penetration.

Fig. 11 Two search trees which illustrate the "penetration" of a search tree. An "elongated" tree has a high penetration, and a "busky" tree has low penetration.

called the "penetration" of a search, or partial search, and may be thought of as representing the degree to which the search tree is elongated rather than "bushy" (see Fig. 11).

4.3 SEARCH TREE PRUNING

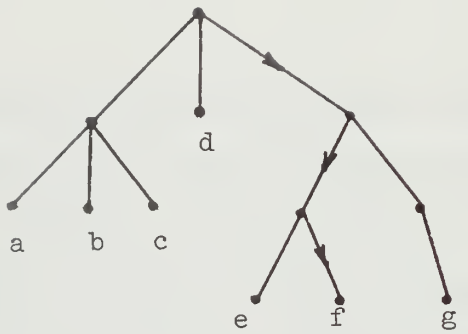
There is a limit to the size which the search tree of the automaton A_0 can attain. Either this will be fixed by the storage capacity of the automaton upon which the search tree program is running, or else it will be determined by the time taken to work with a large tree. The reaction of the program when all the available storage space (about 10 to 30 nodes) has been filled by the tree is described as the partial path is fixed upon, all the tree except a single node is erased, and a new partial search begins (this scheme is illustrated in Fig. 12 (a)). This is unnecessarily catastrophic. There is no need to throw away so much hard-won information and it seems unreasonable to do so.

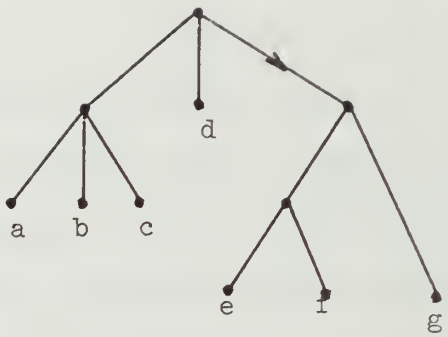
Another version of pruning commits itself only to a single additional step along the path (or some specified number of steps : 1 to 4), and only that part of the tree thereby rendered valueless is discarded. This scheme is illustrated in Fig. 12 (b).

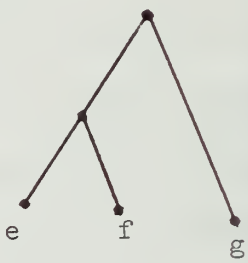
Of course, it is easy to imagine more complicated pruning schemes, e.g., It would be possible to detect high-valued branches without committing the program to any additional path segment, however, we use the second approach of search tree pruning of developed nodes about 10 to 30, while [5] used 40.

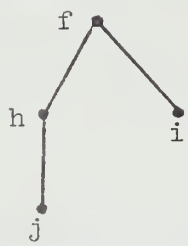
4.4 EVALUATION FUNCTION AS A DISTANCE EVALUATOR

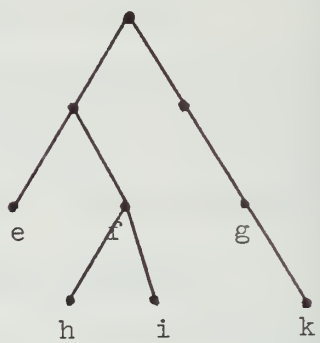
An evaluation function estimates the distance from a state to the goal state. Let us now define a distance estimator as a function which when applied to any two states, specifically estimates the distance over the provided space from the first to the second. Notice that we may discuss good or bad distance estimators for a particular state space (or graph) quite independently of any











(a). static tree.

(b). dynamic tree.

Fig. 12 Illustration of the operation of static and dynamic search trees.

problem associated with that space. Nevertheless, given a good distance estimator, a fortiori we have a good evaluation function for use in any particular search over space.

Now we define the distance estimator as a linear convex sum of m heuristic functions,

$$H(\alpha_i,\alpha_t) = \sum_{k=1}^{m} w_k h_k(\alpha_i,\alpha_t)$$

$$\sum_{k=1}^{m} w_k = 1$$

where $h_k(\alpha_i,\alpha_t)$ is the kth function based upon heuristic grounds between α_i and α_t in space C (examples of evaluation function are shown in the next section), $w_k = \Psi(\alpha_i, \Omega_0, C)$ is the weight for the kth heuristic function and is a function of the present state α_i of the automaton A_0 in space C. In general, the weight w_k for $h_k(\alpha_i, \alpha_t)$ is determined by A_0 considering the local and grobal environment at the state α_i.

The idea of the linear convex sum of some heuristic functions is intuitive since it is almost always impossible to find a quite simple single function which can be applied to any given geometrical space and works effectively at any state of the given space. In other words, some functions are efficient for certain geometrical configuration while some others may be very inefficient for such configuration. Our distance estimator $H(\alpha_i, \alpha_t)$ can be interpreted as a simple structure of the reticular formation in animals [39].

In the previous sections we discussed several trees whose devlopment is so called "uni-directional." According to our assumption that the given automaton A_0 knows the picture of the world C, we consider a version of so-called "bi-directional" search tree [31], which may be called "rendezuous." This is an approach that from both present state α_i and A_0 and the

present state β_i of the image of A_0, denoted by A_0' of Ω_0', the automaton develops two search trees. One towards β_i from α_i and the other from β_i to α_i, and try to intersect Ω_0 with the image Ω_0'.

Let α_i and β_i be the ith states of A_0 and A_0', respectively. Then our distance evaluator may be modified to:

$$H(\alpha_i, \beta_i) = \sum_{k=1}^{m} w_k h_k(\alpha_i, \beta_i)$$

where $$w_k = \varphi(\alpha_i, \beta_i, \Omega_0, \Omega_0', C) \text{ , for all k.}$$

The decision process of which search tree should be developed at ith stage depends on the local environments of the automaton and the image of itself, and it is not specifically discussed in this paper.

4.5 EXAMPLES OF DISTANCE EVALUATION FUNCTION

It is desirable to consider a function to evaluate the distance between α_i and β_i with straight line segments as the one illustrated in Fig. 13. However, this may not be efficient on computational aspect unless very fast algorithm for such a scheme is easily implemented. In fact, if C consists of many obstacles and if large numbers of obstacles are located around α_i and β_i, it may quite often lead to a very inefficient procedure to estimate distance between α_i and β_i. Since we may not strictly require the automaton A_0 to take one of the optimal paths if any exists, but rather hope that it may take a path which is close to an optimal path as well as the path seeking time is small, we may have some functions which are not quite like the one in Fig. 13, but yet they are easy to implement as well as in some sense "good" evaluation functions.

Some of simple heuristic distance estimation functions for given two states α_i and β_i are illustrated in Figures 14, 15, 16, 17, and 18. For example, $h_1(\alpha_i, \beta_i)$ function considers those obstacles which intersect the line segment

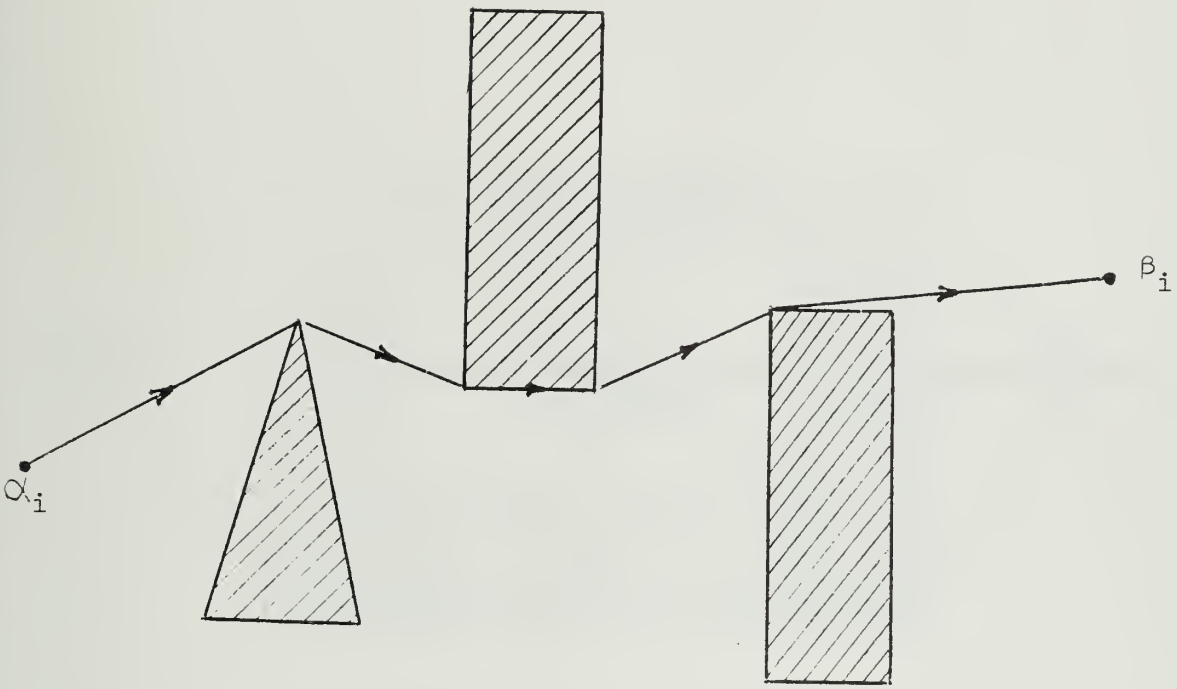


Fig. 13 Distance estimator with straight line segments.

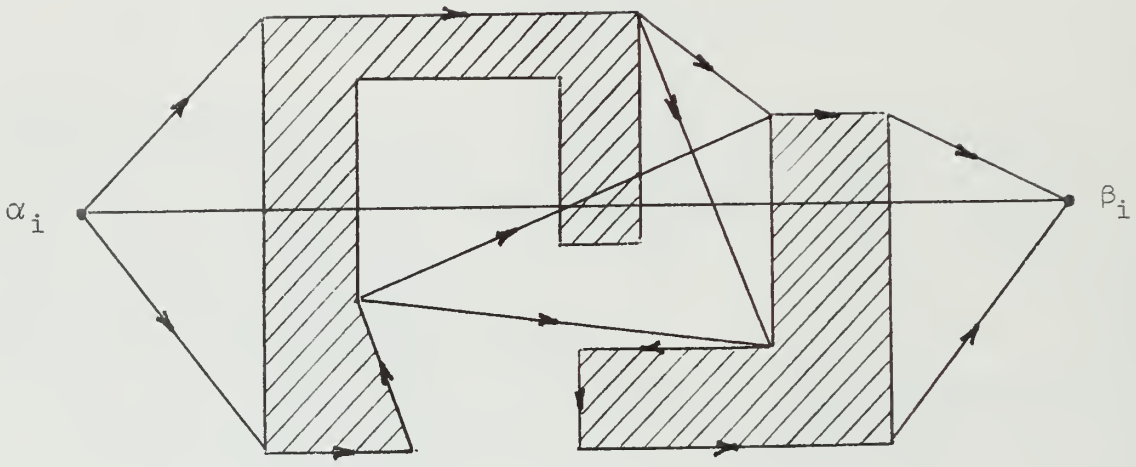


Fig. 14 Distance evaluation function $h_1(\alpha_i, \beta_i)$, around those obstacles on the line segment.

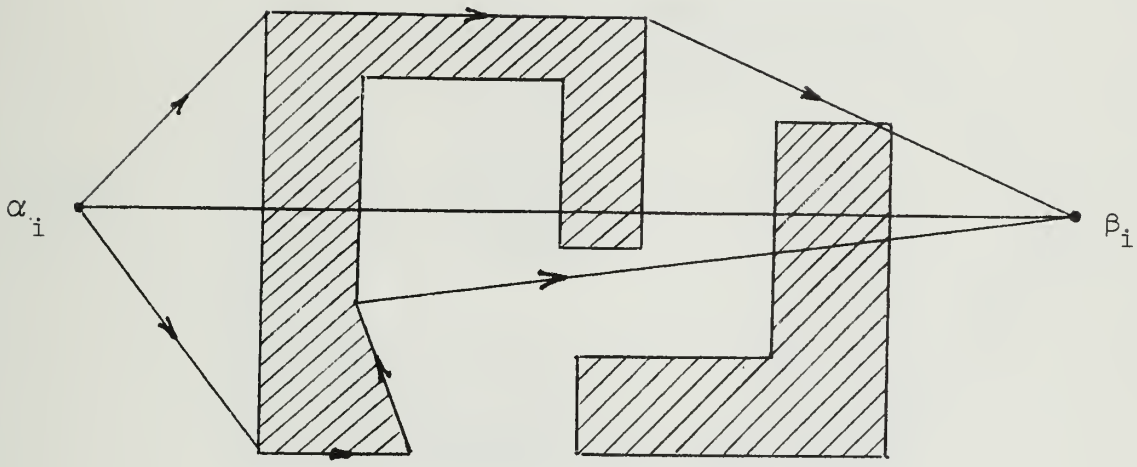


Fig. 15 Distance evaluation function $h_2(\alpha_i, \beta_i)$, around only the nearest obstacle to α_i.

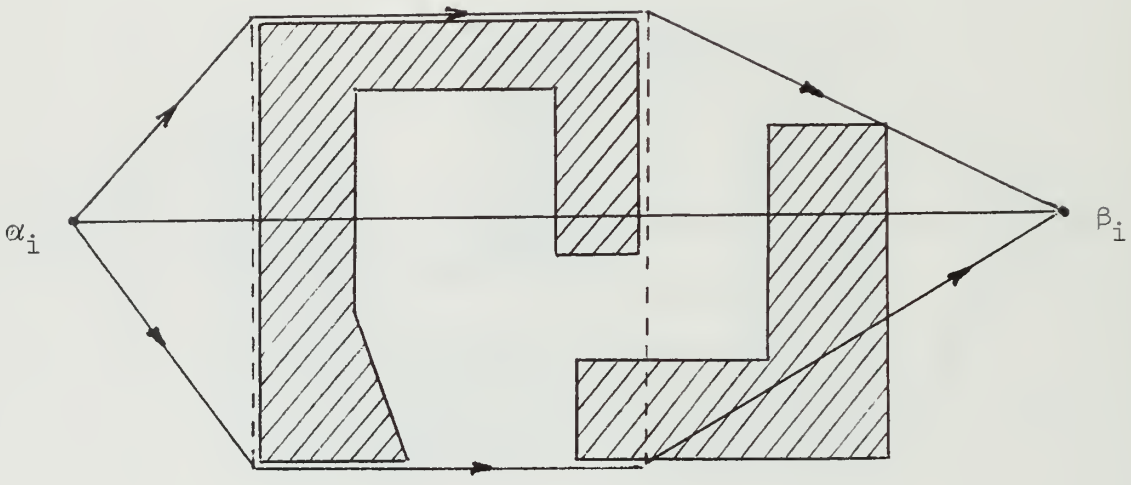


Fig. 16 Distance evaluation function $h_3(\alpha_i, \beta_i)$, around only the minimal box of the nearest obstacle to α_i.

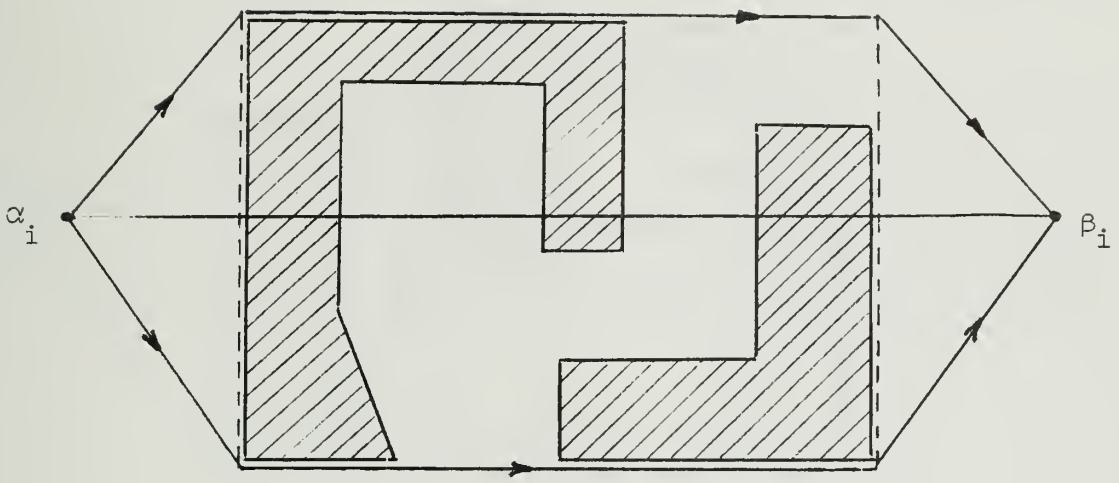


Fig. 17 Distance evaluation function $h_4(\alpha_i, \beta_i)$, around the union of the minimal boxes of those obstacles on the line segment.

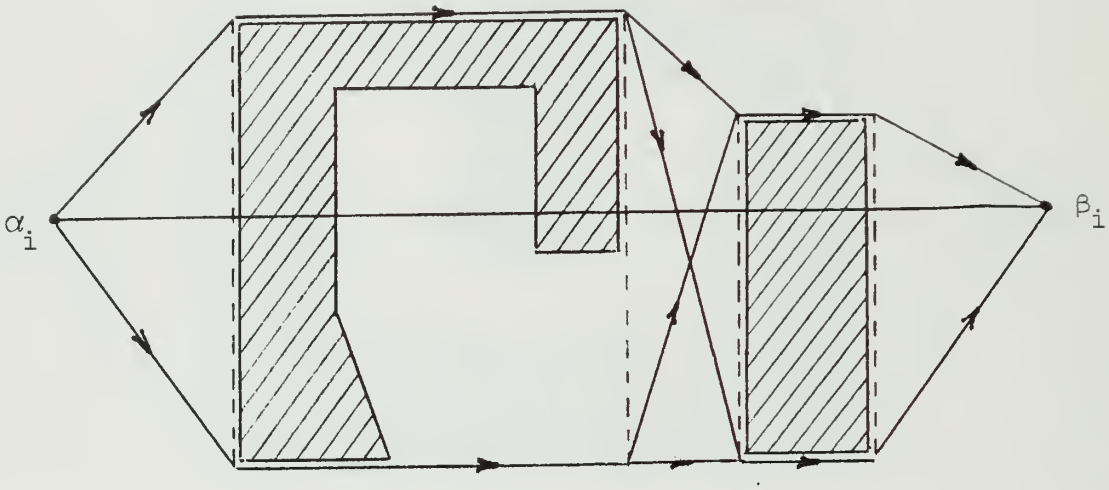


Fig. 18 Distance evaluation function $h_5(\alpha_i, \beta_i)$, around the minimal boxes of these obstacles on the line segment.

$\overline{\alpha_i,\beta_i}$ and take the direction corresponds to the minimum path of all 2^M paths, where M is the number of obstacles on the line segment. The function $h_2(\alpha_i, \beta_i)$ considers only one obstacle which is nearest to the state α_i and take the direction of the minimum of two possible paths, and so on for h_3, h_4, and h_5. Of course, there may be some other functions other than h_1 through h_5 which can be easily implemented. Some of these illustrated distance estimation functions may be ill-defined depending upon where α_i and β_i are located with respect to these obstacles on the line segment $\overline{\alpha_i,\beta_i}$.

A procedure called BSA, bullet-shooting algorithm, is implemented in the automaton A_0 to detect the obstacles on the line segment $\overline{\alpha_i,\beta_i}$ as well as the first obstacle which is hit by the bullet, if there exist any obstacles on the segment. A bullet of length δ is shot from α_i toward β_i on the line segment,

$$x = (x_i' - x_i)T + x_i$$

$$y = (y_i' - y_i)T + y_i$$

$$0 \leq T \leq 1$$

$$\alpha_i = (x_i, y_i)$$

$$\beta_i = (x_i', y_i')$$

with a step motion V, $0 < V < \delta$, and detect intersection between the bullet and obstacles, if any are on the line segment. If there are no obstacles on the line segment, the automaton moves toward β_i.

When the automaton is located in a region with many obstacles, called "bush", then it may be desirable for the automaton A_0 to have a decision process whether A_0 should first get away from such a bush and then seek for his image state β_i rather than seeking the image through the bush. Which means it is desirable for him to have an ability to decide whether the bush is local or global.

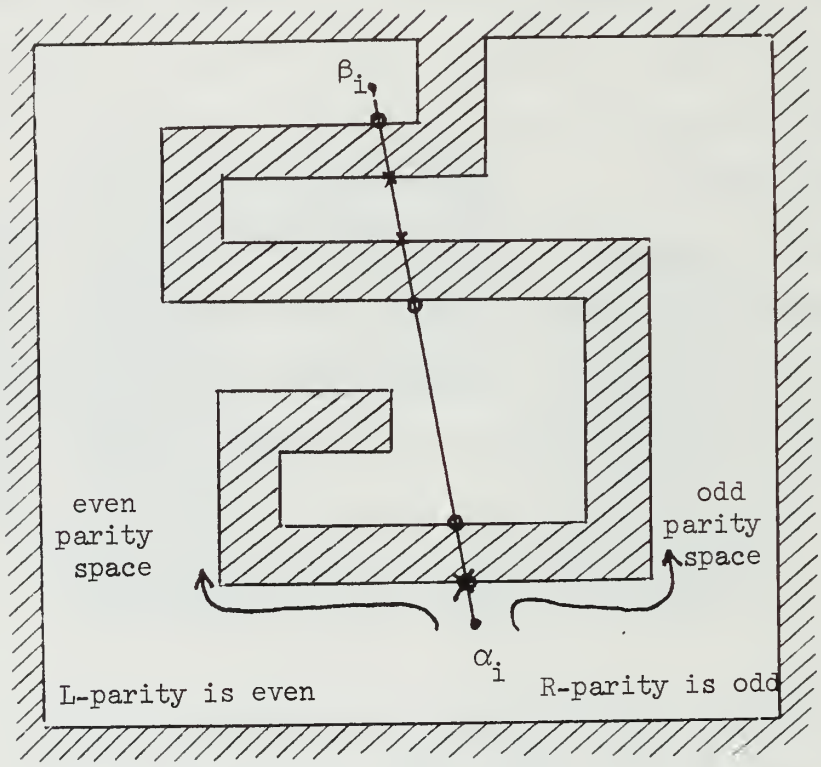


Fig. 19 Direction parities: L-parity and R-parity which assume either even or odd.

Let us assume an automaton is located in the interior of an obstacle, this situation is illustrated in Fig. 19. Then, sometimes or quite often, it takes direction to seek the goal in which it never achieves the goal until after such a space is almost exhaustively searched and then it realizes the direction which has been taken before was wrong. For example, in Fig. 26, the automaton never achieves the goal if he started to look for the goal in the right space with respect to the line $\overline{\alpha_i,\beta_i}$. This situation can be easily avoided by the following approach.

For the left and the right direction from α_i toward β_i, we define direction parities as the number of intersection of edges on the line segment $\overline{\alpha_i,\beta_i}$. L-parity is found by the left edge following and R-parity is found by right edge following. The L-parity and R-parity assume either even or odd values, thus the automaton searches for the goal in the space whose parity is even. This condition is illustrated in Fig. 19.

4.6 SIMULATION RESULTS OF GOAL SEEKING AUTOMATON

In this section, we present some computational results of our goal seeking automaton A_0 of shape Ω_0 under the assumption that A_0 has complete information of the world C. Ω_0 of A_0 and obstacles Ω_i, i=1, ..., M are assumed to be polygons.

Transformations provided for A_0 are R and either T_4 or T_8:

$$T_4 = \{t_1, t_2, t_3, t_4\}$$

$$T_8 = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$$

$$R = \{r:n\}$$

where T_4 and T_8 are two sets of translations, and R consists of rotations. The clockwise rotation of Ω_0 at its center point $\overline{p}$ is denoted by positive r where $r = \pi/n$ and the counter clockwise rotation by negative r. T_4 and T_8 are illustrated in Fig. 20.

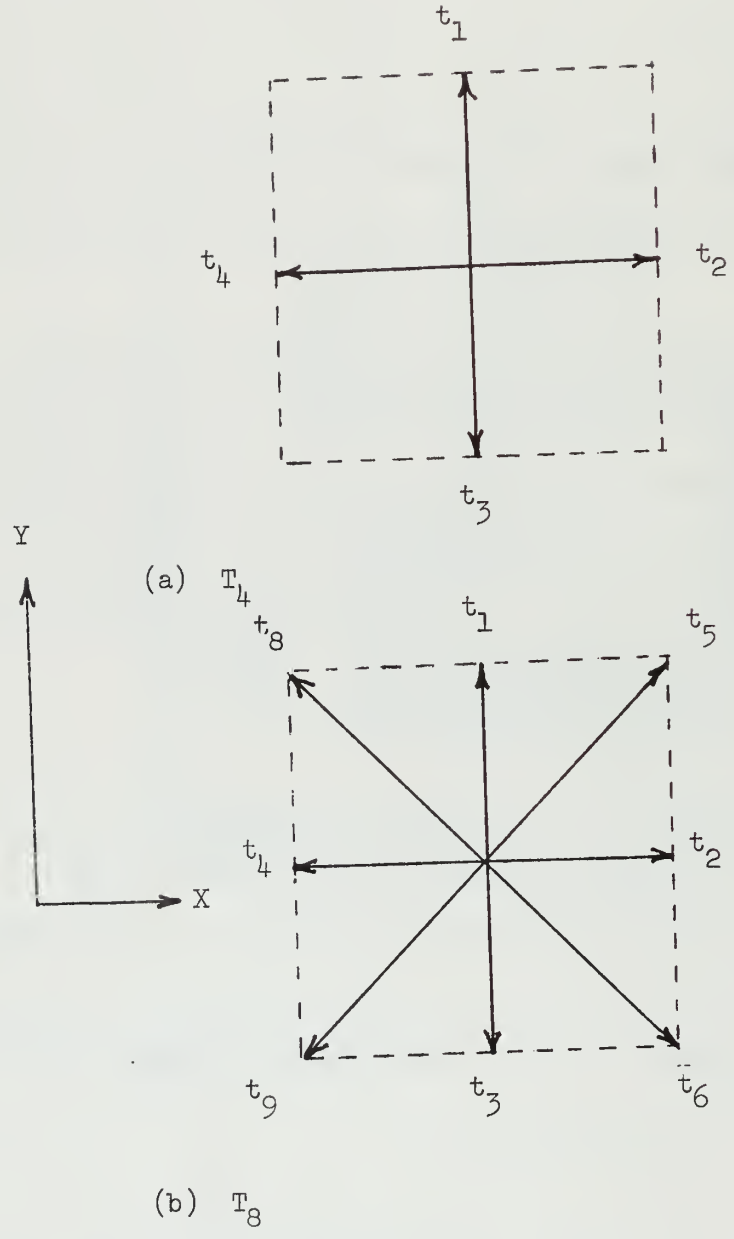


Fig. 20 Sets of translations: T_4 and T_8.

Let $\Omega_0(P, E; \bar{p})$ and $\Omega_0'(P', E'; \bar{p}')$ denote the polygons of the automaton A_0 and its image A_0' from the goal state, respectively. In general, we have the following linear convex sum function as a distance evaluator.

$$H(\bar{p},\bar{p}') = w_{-1}d(\bar{p},\bar{p}') + w_0 h_j(\bar{p},\bar{p}') + (1/n) \sum_{k=1}^{m} \sum_{i=1}^{n} w_k h_k(p_i,p_i')$$

$$p_i \epsilon P,\ p_i' \epsilon P',\ |P| = |P'| = n$$

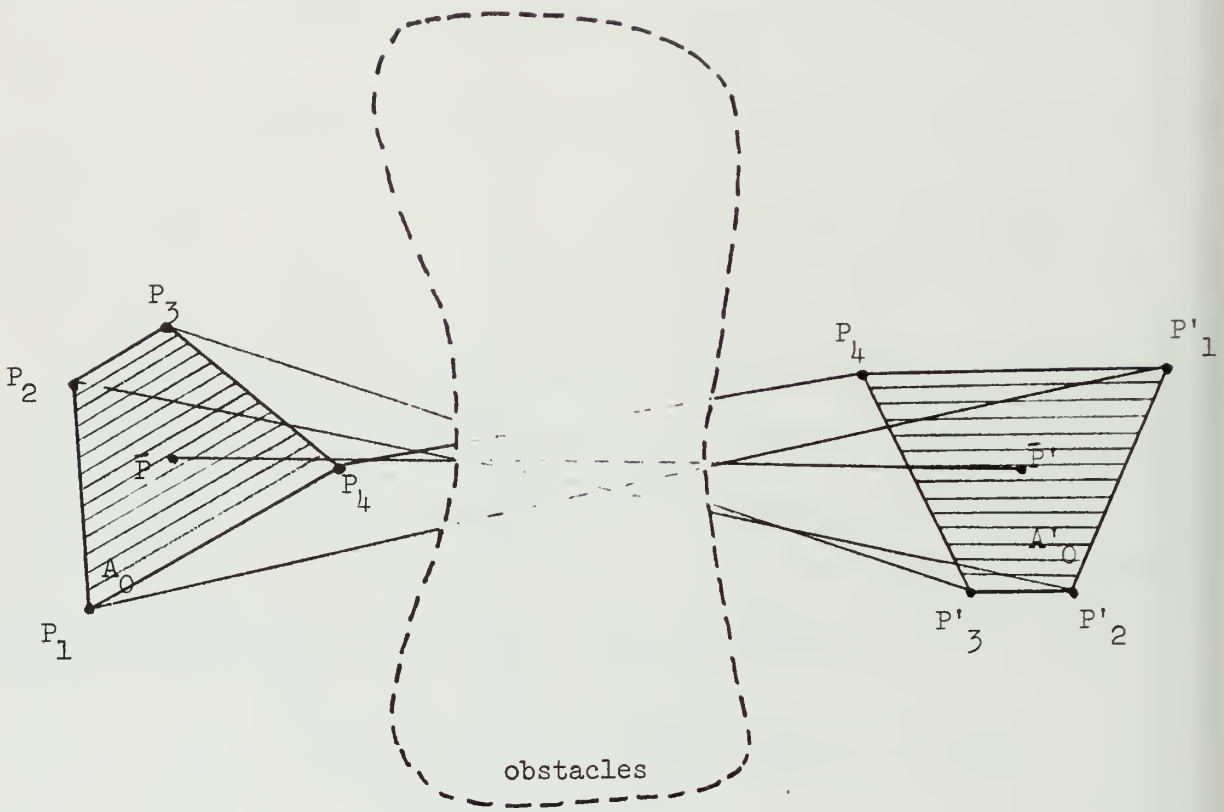
$$w_{-1} + w_0 + (1/n) \sum_{k=1}^{m} \sum_{i=1}^{n} w_k = 1$$

where d denotes Euclidian distance function. Since it is natural to set $d(\bar{p},\bar{p}') = h_j(\bar{p},\bar{p}')$ if there are no obstacles on the line segment $\overline{p.p'}$, and moreover if some obstacles are on the line segment then we want to set $\omega_{-1} = 0$, we implement d function in each h_j, $j = 1, 2, \ldots, m$.

We use the following simplified form of (1) as our distance evaluator

$$H(\bar{p},\bar{p}') = \omega h_k(\bar{p},\bar{p}') + (1 - \omega)/n \sum_{i=1}^{n} h_k(p_i,p_i') \qquad (2)$$

This evaluation function can be interpreted as follows. The first term contributes mainly to translation of Ω_0 while the second term contributes to the rotation of Ω_0. Thus the contribution to either translation of Ω_0 or to rotation of Ω_0 is controlled by the value of ω, $0 \leq \omega \leq 1$. If ω is small then rotational transformations may be applied on Ω_0 in the earlier stages of path seeking and later translations may be applied on Ω_0. On the other hand, if ω is quite large then translations are considered more important than the rotation, thus we may inspect that rotations of Ω_0 will be remained until the end of path seeking. It is also important to notice that the second term in (2) also contributes to translations of Ω_0. If $\omega = 1$ for example, Ω_0 is translated into the goal with no relational transformation, while for $\omega = 0$ both rotation and translations are applied on Ω_0 to lead to the goal.



$$H(\bar{p}, \bar{p}') = wh(\bar{p}, \bar{p}') + ((1-w)/4)\sum_{i=1}^{4} h(p_i, p_i')$$

Fig. 21 Distance evaluator $H(\bar{p}, \bar{p}')$.

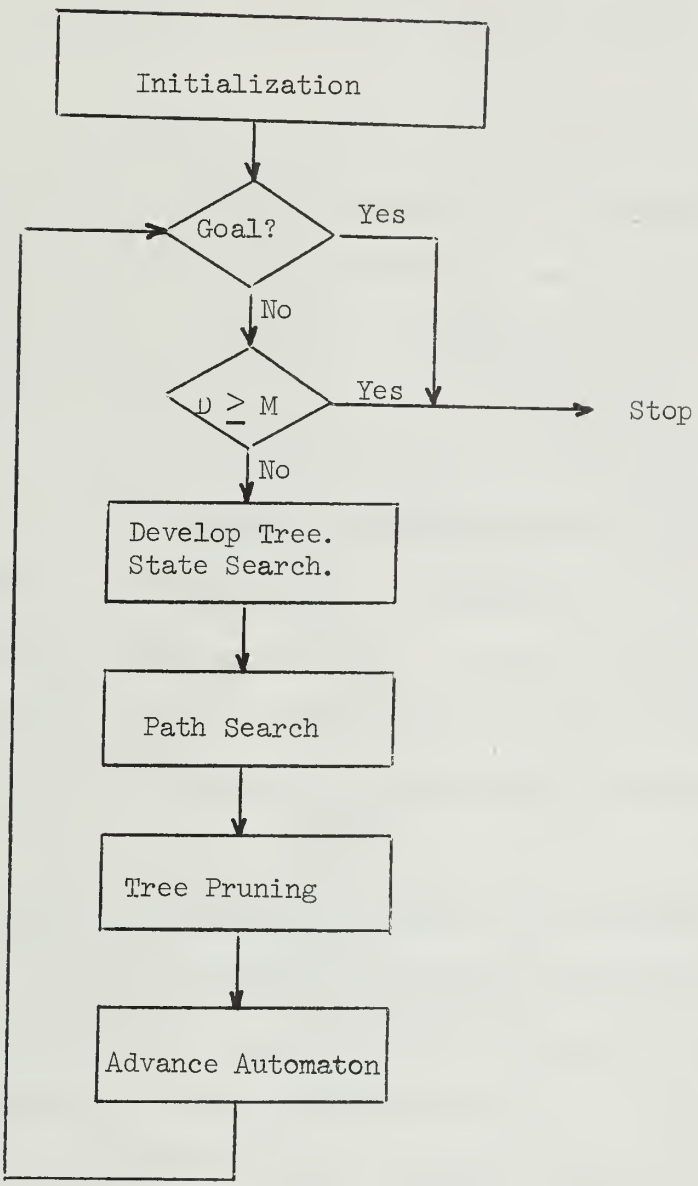


Fig. 22 Diagram of the automaton A_0.

A simple example of this distance evaluator $H(\overline{p},\overline{p}')$ is illustrated in Fig. 21.

We describe a simplified diagram of our goal seeking automaton A_0 in Fig. 22, where the function of each block is almost as same as discussed in the previous sections.

Under the above assumption, we let our automaton A_0 of Ω_0 seek the specified goal, denoted by P_t, from another specified starting position P_s in various world C. Some of the results obtained by A_0 are illustrated in Fig. 23 through Fig. 36. Fig. 23, for example, the world C consists of two obstacles Ω_1 and Ω_2 which are bounded by Ω_3. Ω_0 of A_0 is a triangle and h_k of $H(\overline{p},\overline{p}')$ is simply chosen as h_2 of Fig. 15, and the weight ω is set to 1 which means the second term of $H(\overline{p},\overline{p}')$ is omitted thus the path attained by A_0 consists of only translations from T_4. Fig. 24 shows the same constraints as the one in Fig. 23 except h_k is chosen as h_1 of Fig. 14, and the provided translations are T_8. As we can see easily that for T_8 the seeked path is smoother than the one achieved by T_4, and the choice of h_1 eliminates the "zig-zag" walk which happened by h_2 in Fig. 23.

Fig. 25 through Fig. 33 show the results under the similar assumptions applied in cases of Fig. 23 and Fig. 24 in different world C. In Fig. 34 and Fig. 35, the weight ω is set to less than 1 and rotational transformations $R = \{r;6\}$, are applied to achieve the goal. As we can see from these figures, the change of the value of ω between 0 and 1 indicates that the necessary rotations are distributed on the path sequence and such a distribution of rotations can be controlled by changing the value of ω. Fig. 36 shows the case where $\omega = 0.5$ and rotational transformations, some of them are redundant, are distributed on the seeked path sequence almost evenly.

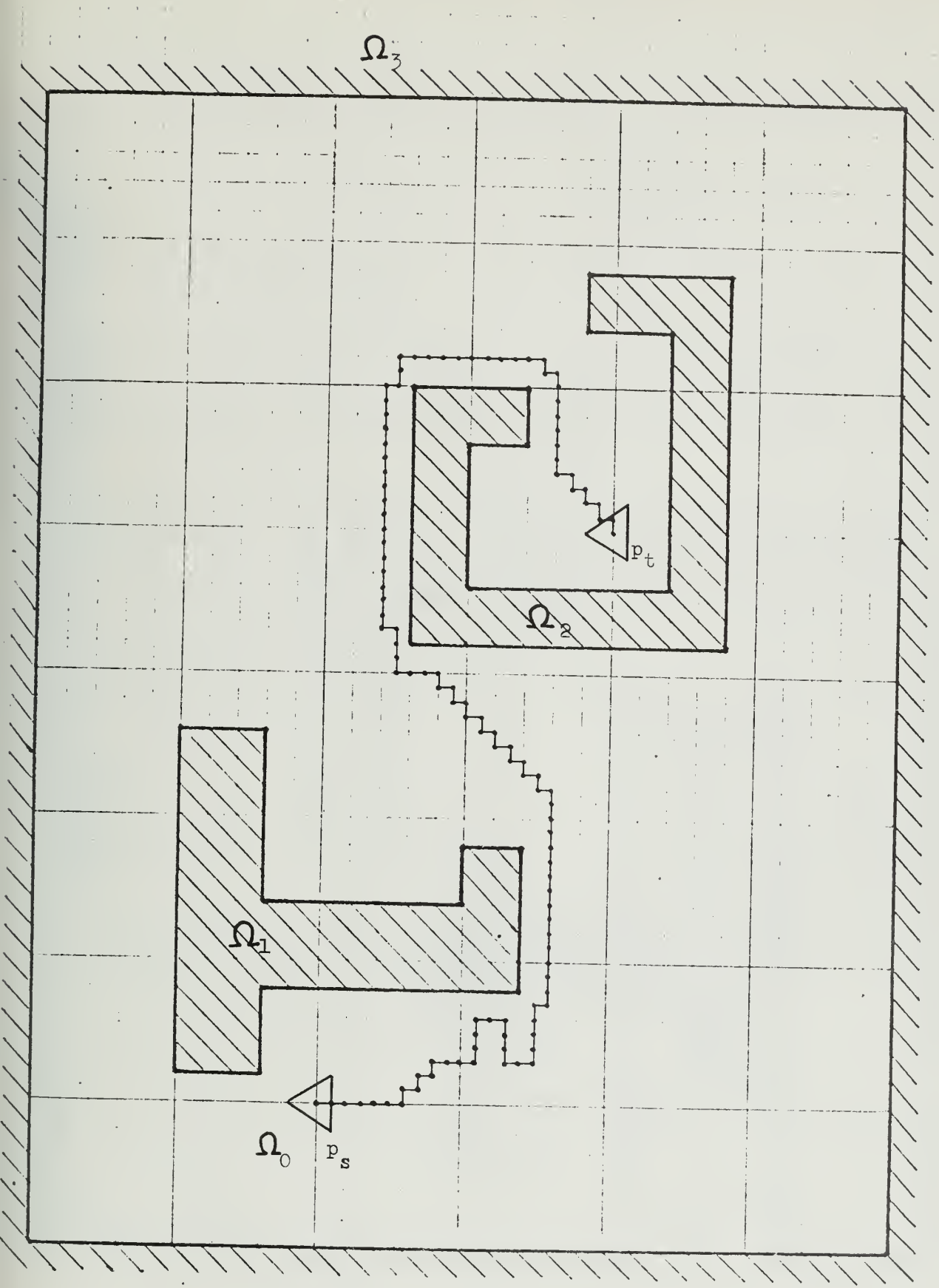


Fig. 23 $h_k = h_2$, $\omega = 1$ and T_4

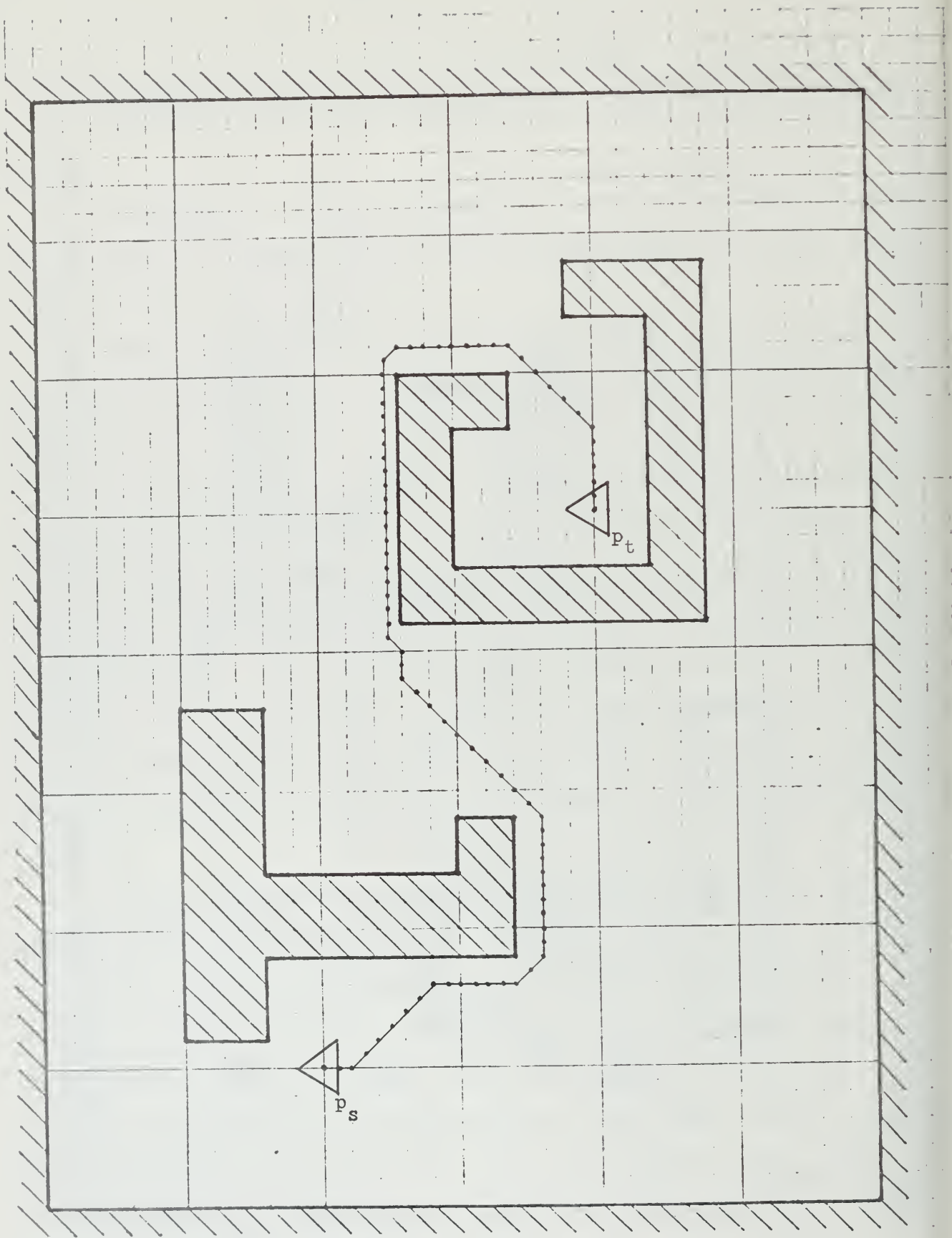


Fig. 24 $h_k = h_l$, $\omega = 1$ and T_8

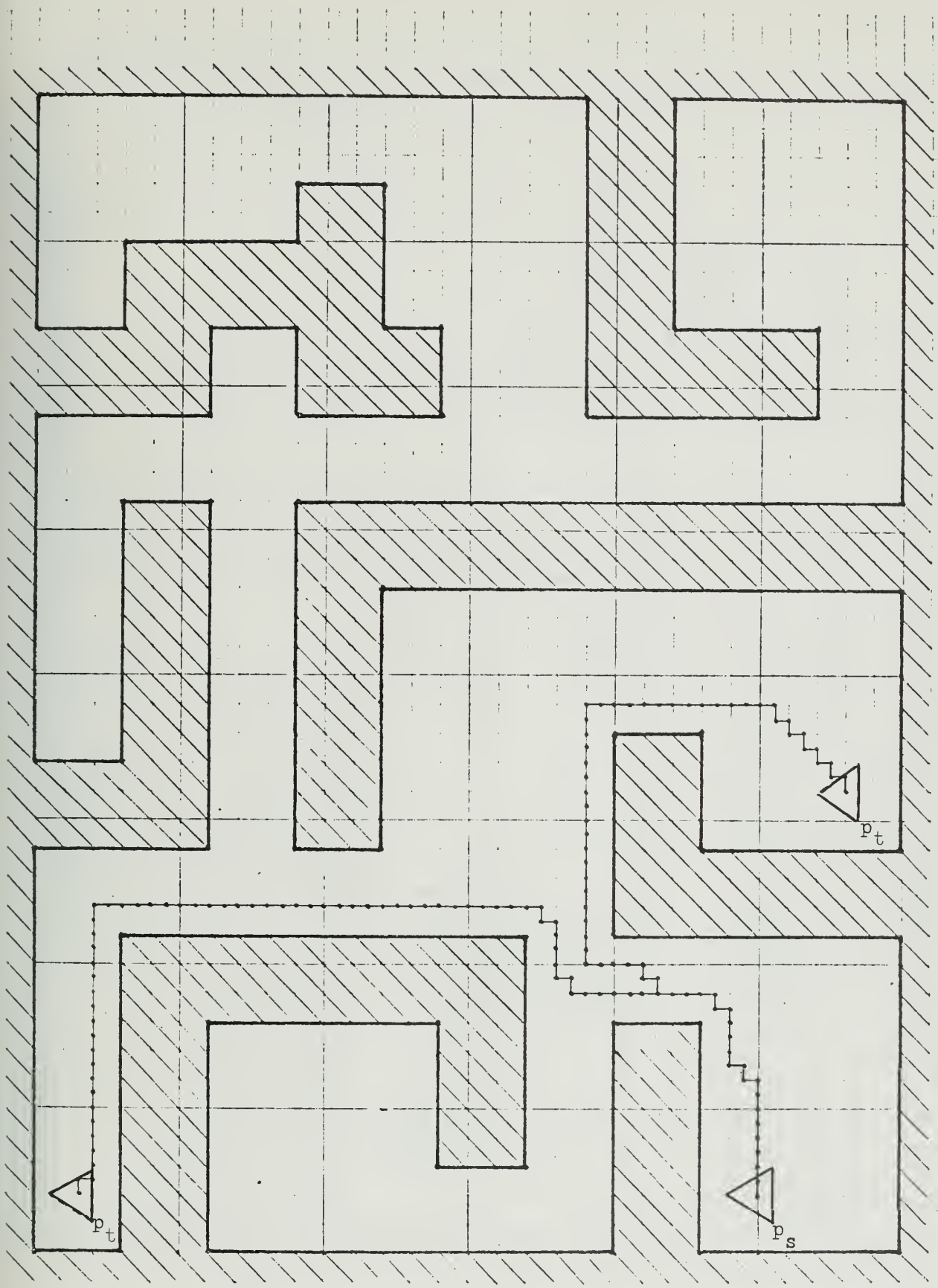


Fig. 25 $h_k = h_2$, $\omega = 1$ and T_4

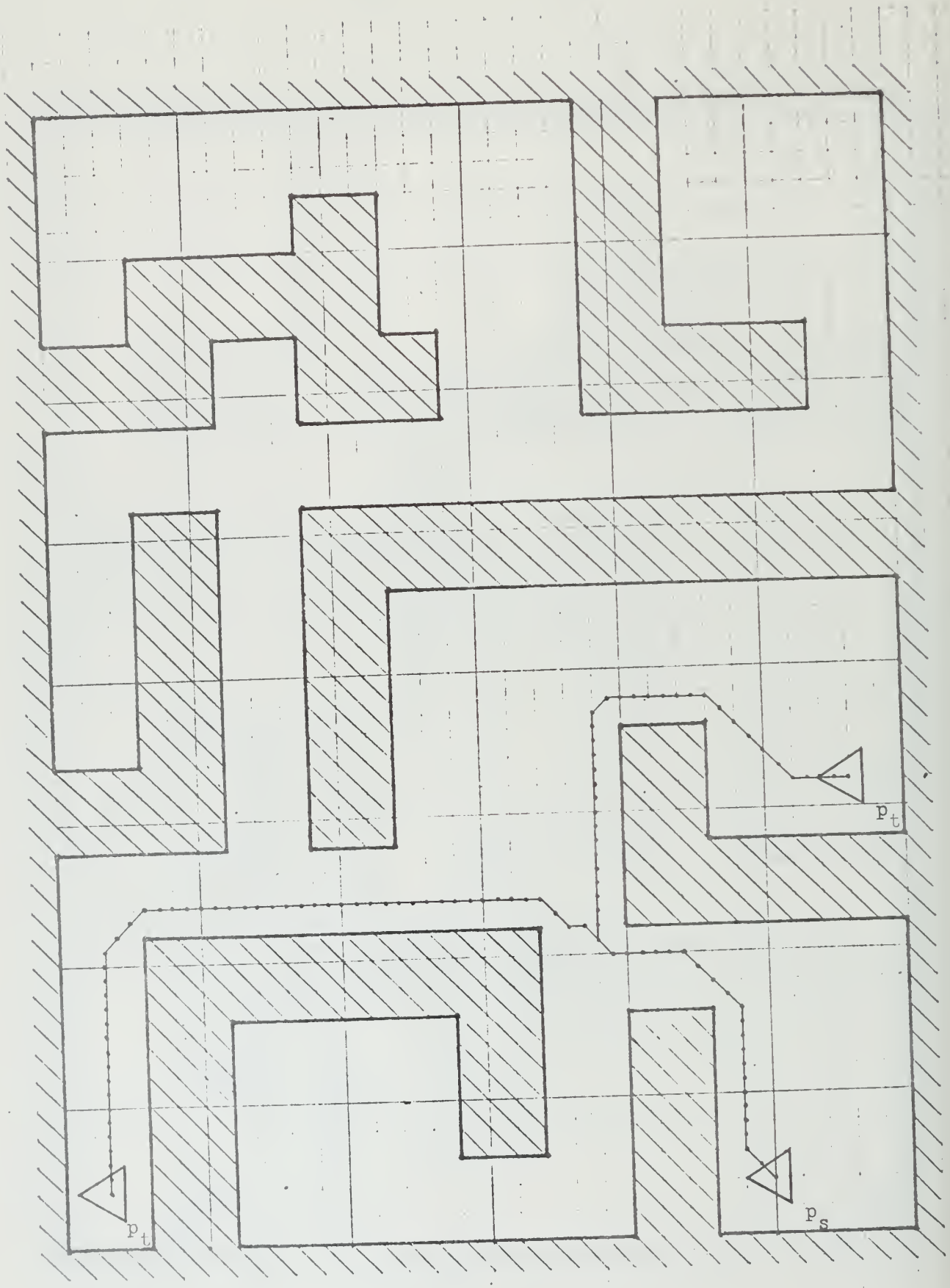


Fig. 25 $h_k = h_2$, $\omega = 1$ and T_8

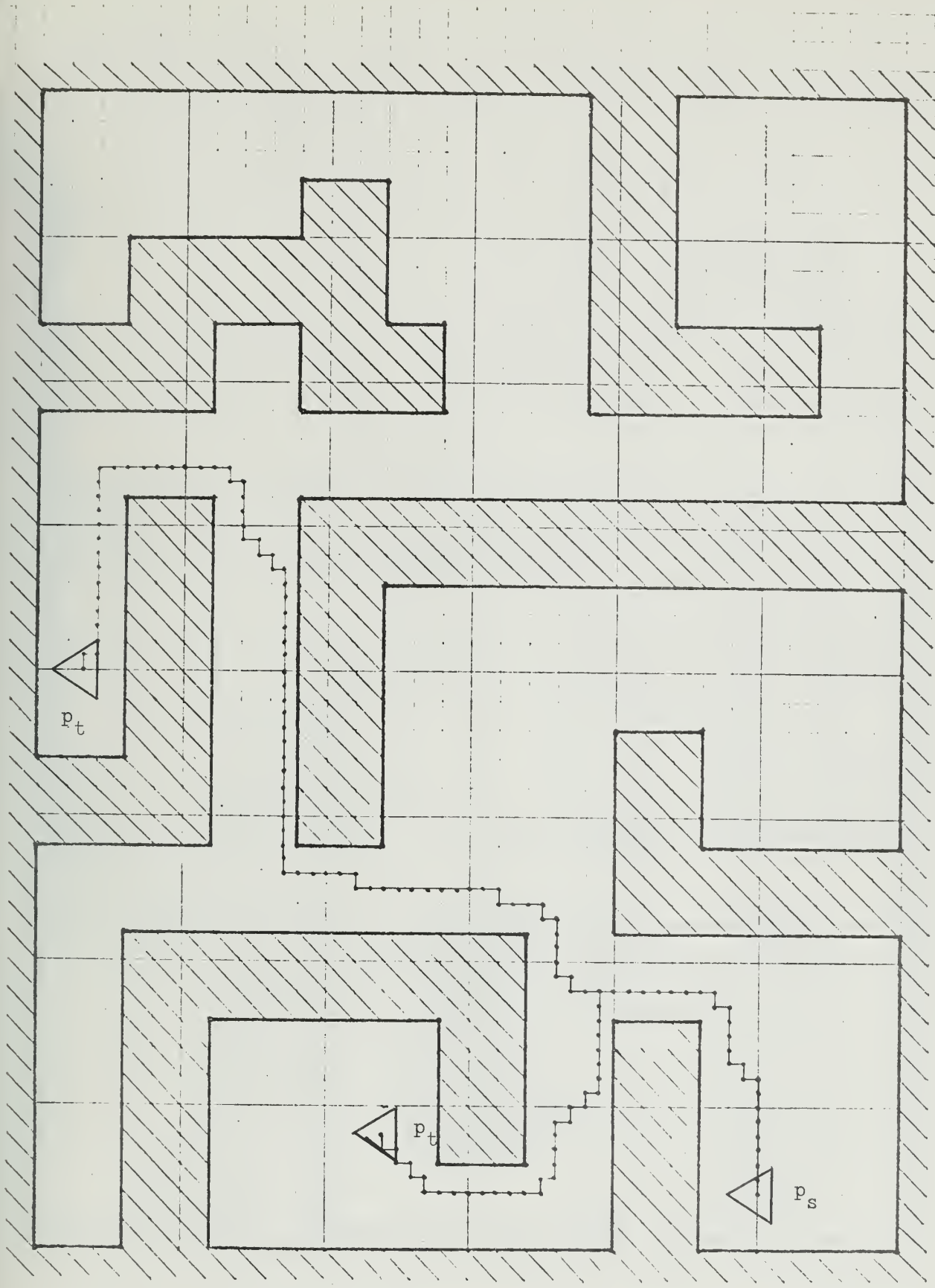


Fig. 26 $h_k = h_2$, $\omega = 1$ and T_4

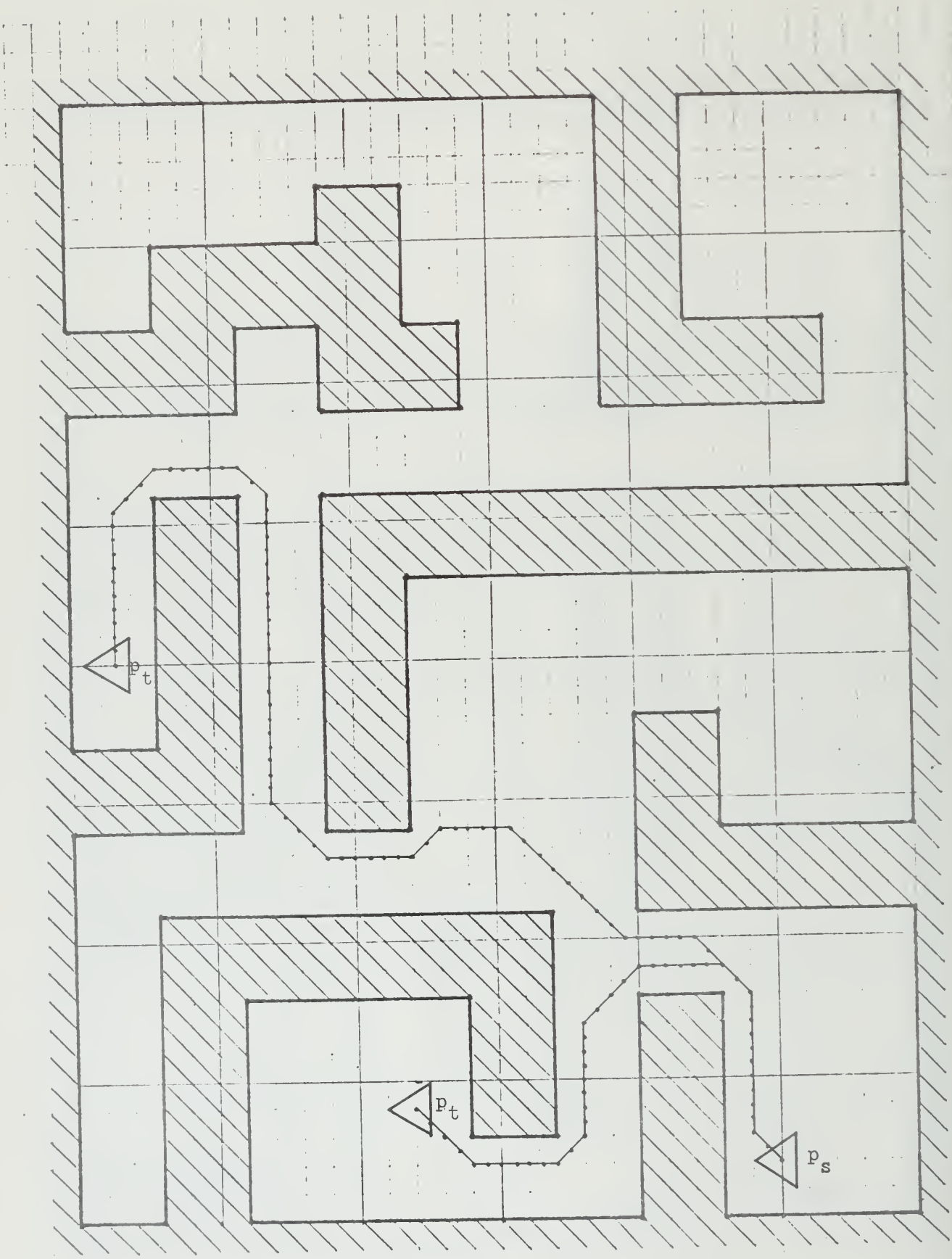


Fig. 27 $h_k = h_2$, $\omega = 1$ and T_8

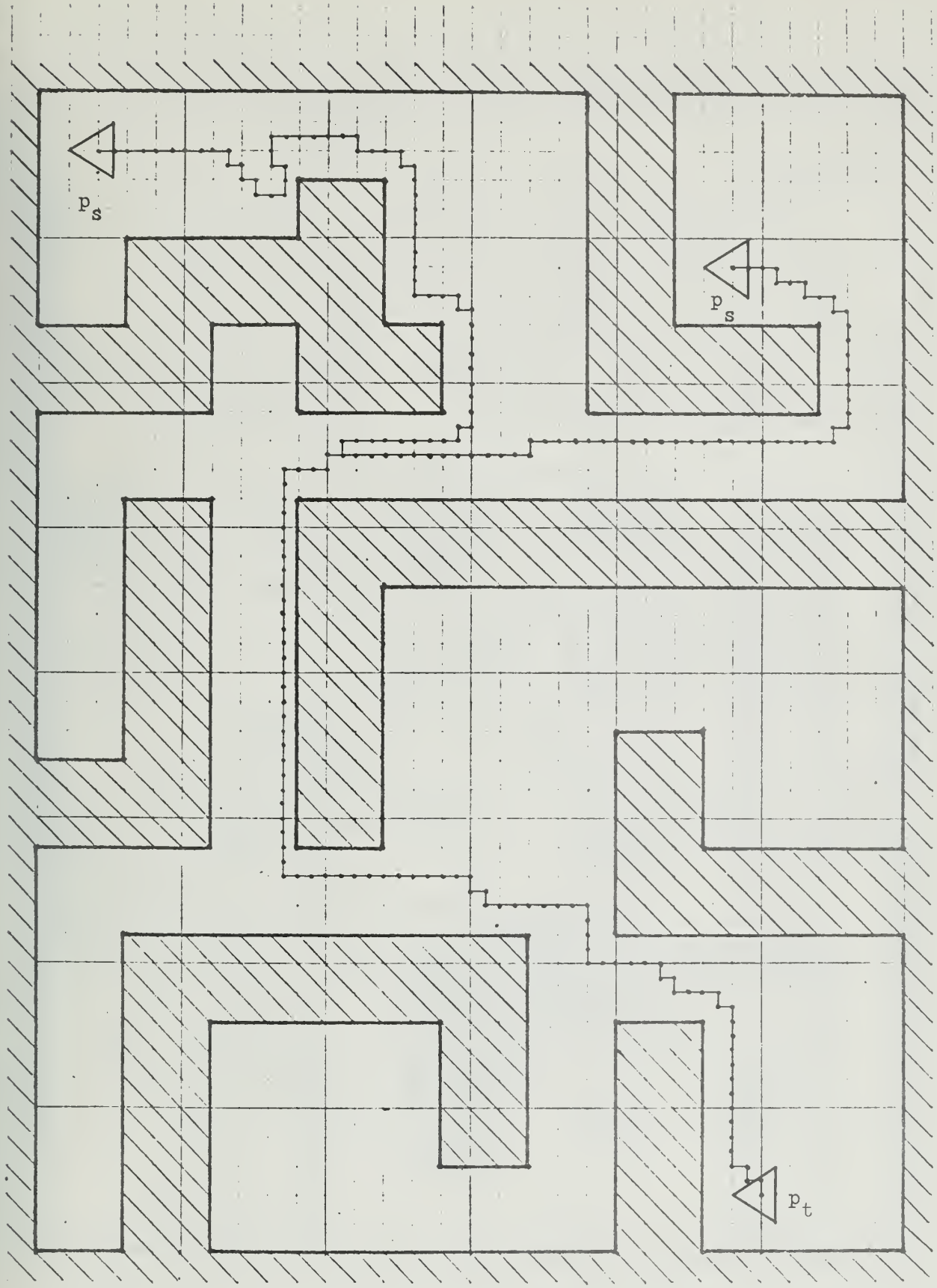


Fig. 28 $h_k = h_2$, $\omega = 1$ and T_4

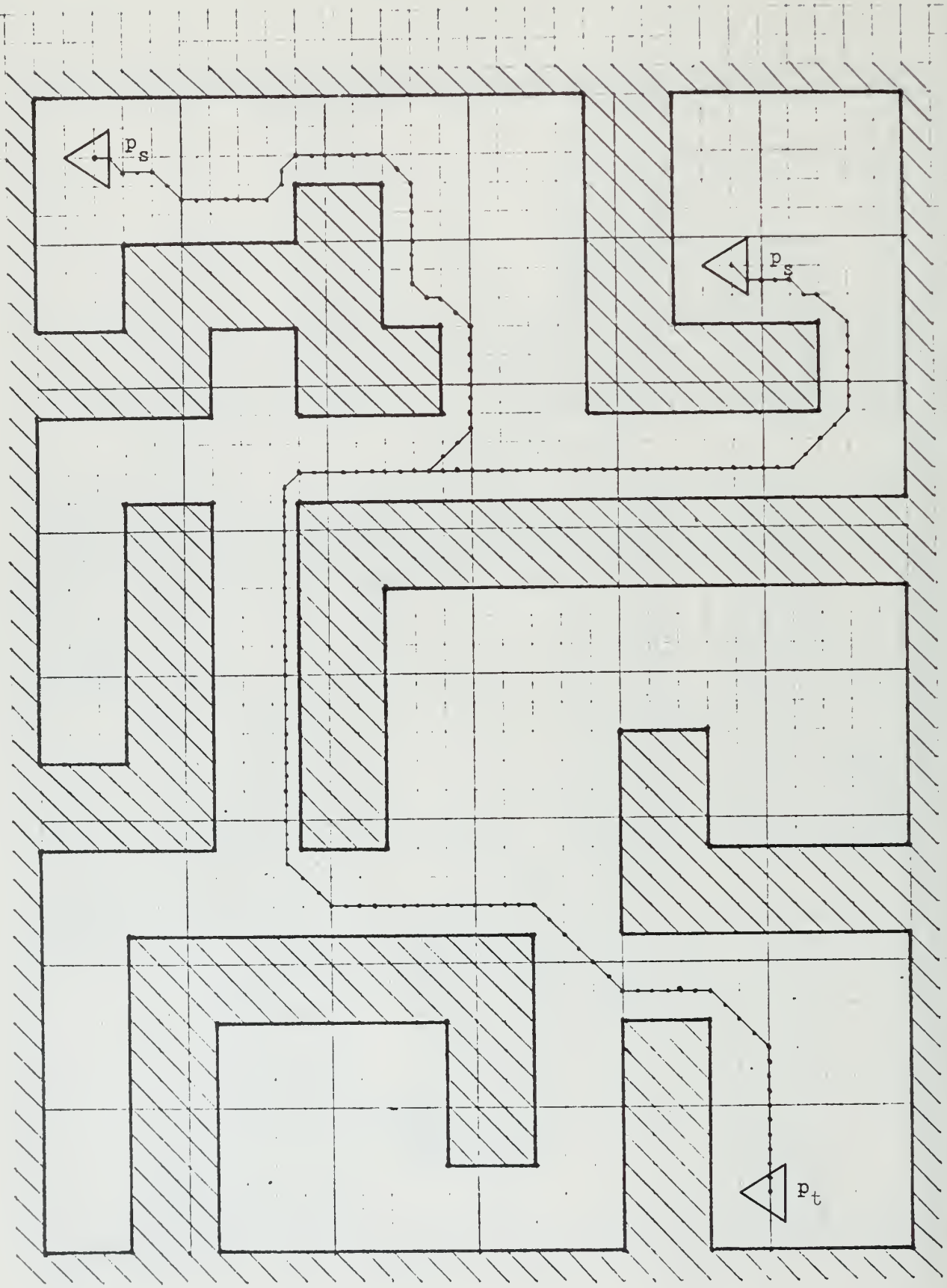


Fig. 29 $h_k = h_2$, $\omega = 1$ and T_8

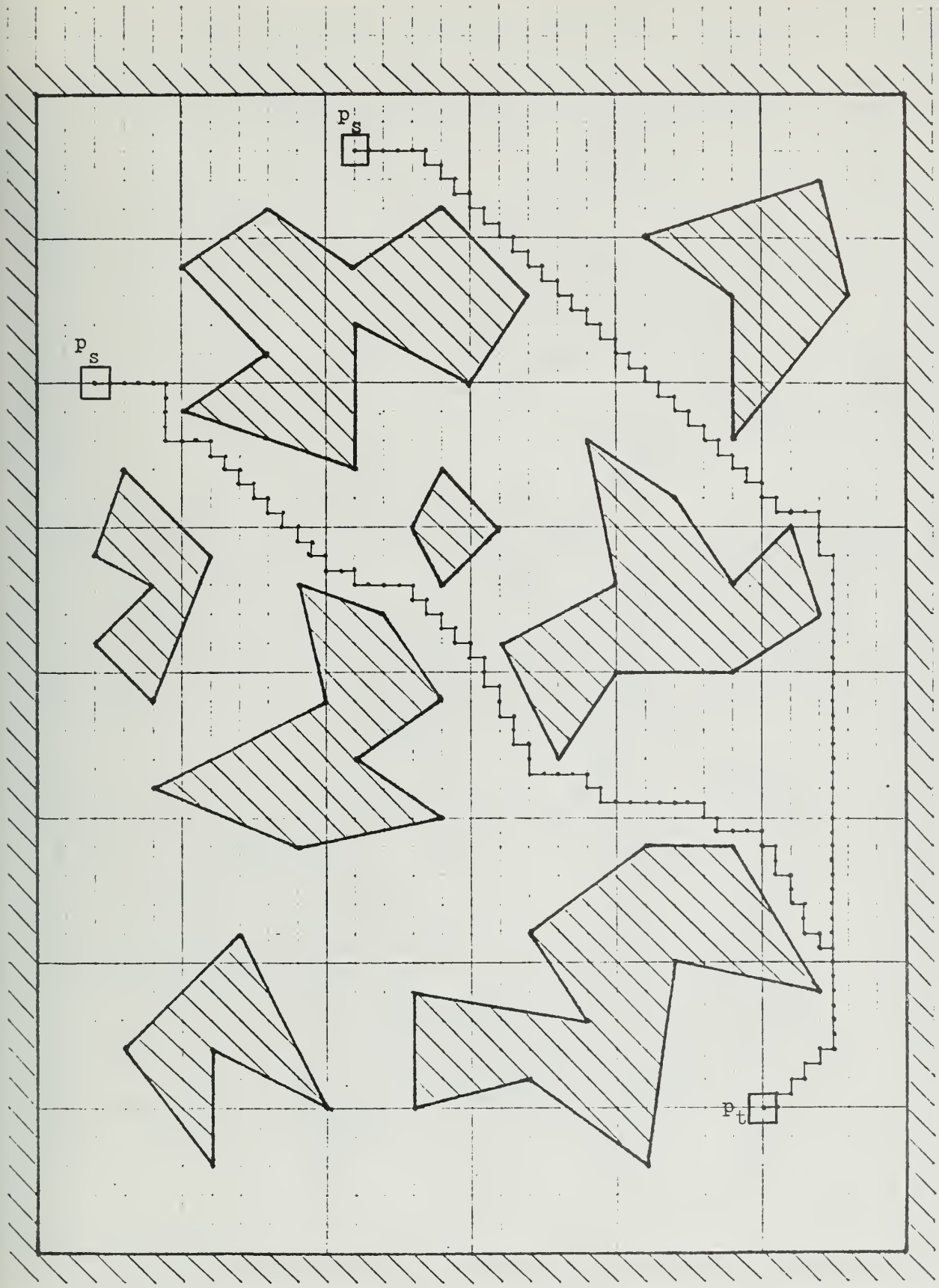


Fig. 3 $h_k = h_2$, $\omega = 1$ and T_4

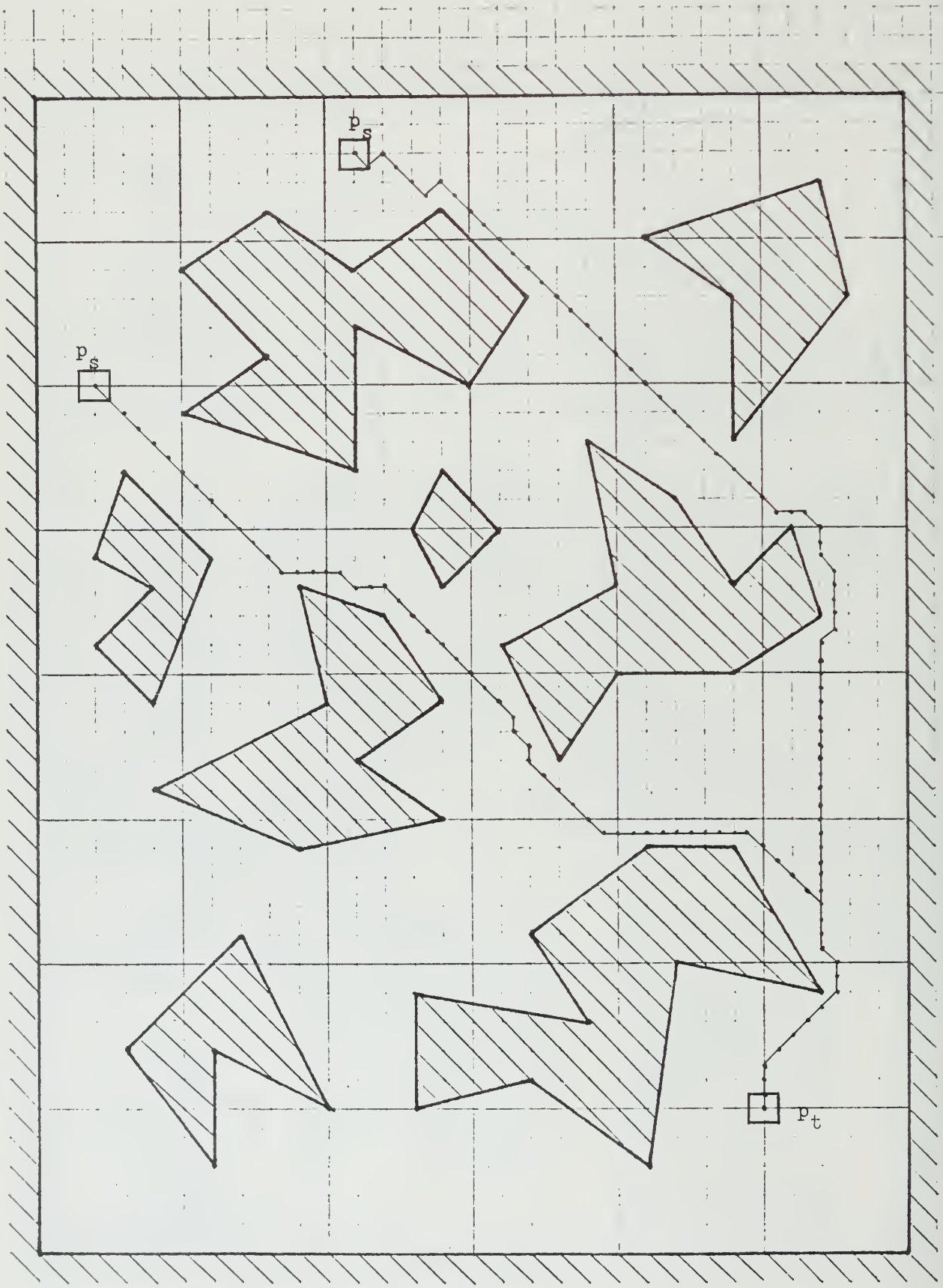


Fig. 31 $h_k = h_2$, $\omega = 1$ and T_8

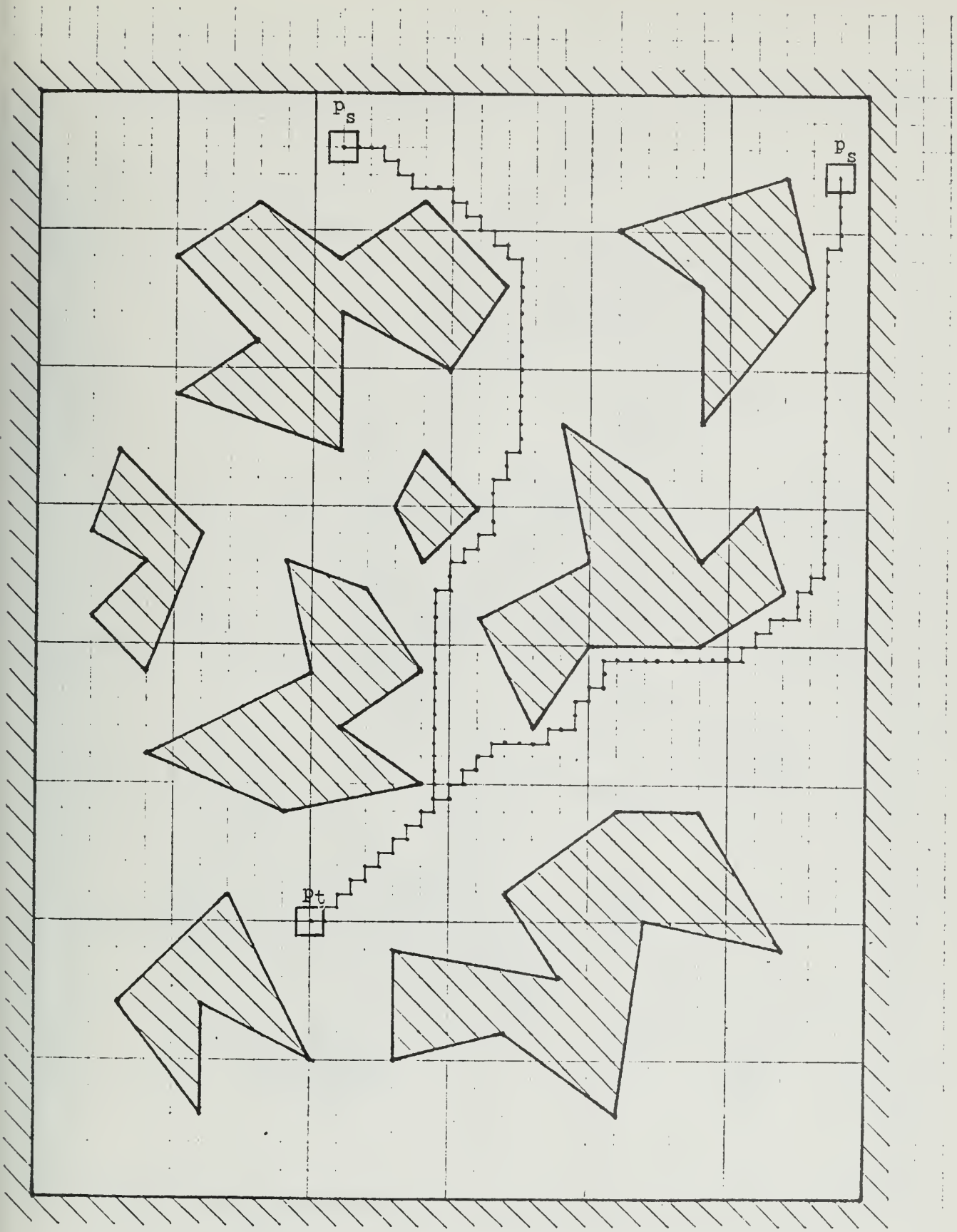


Fig. 32 $h_k = h_2$, $\omega = 1$ and T_4

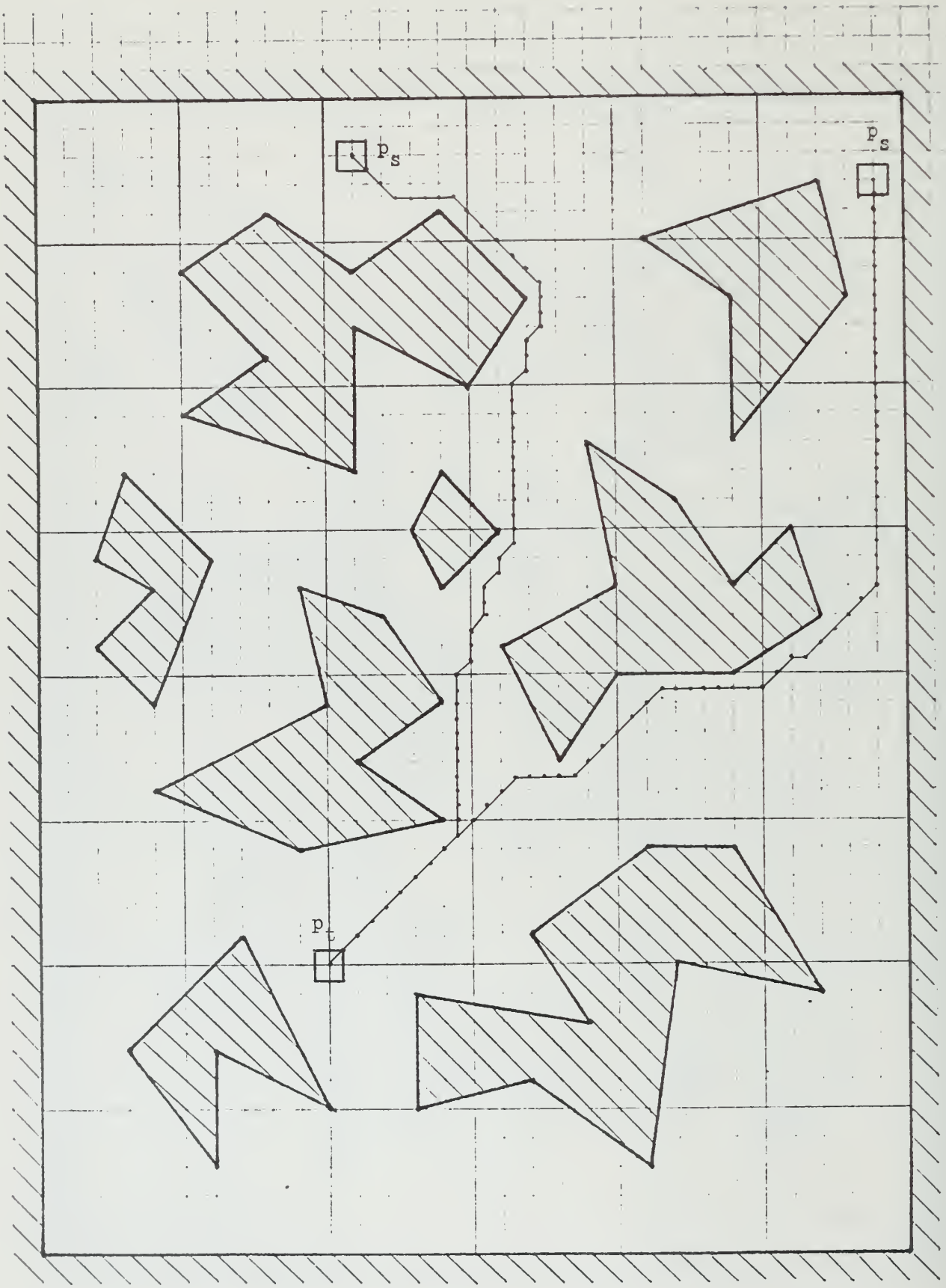


Fig. 33 $h_k = h_1$, $\omega = 1$, T_8

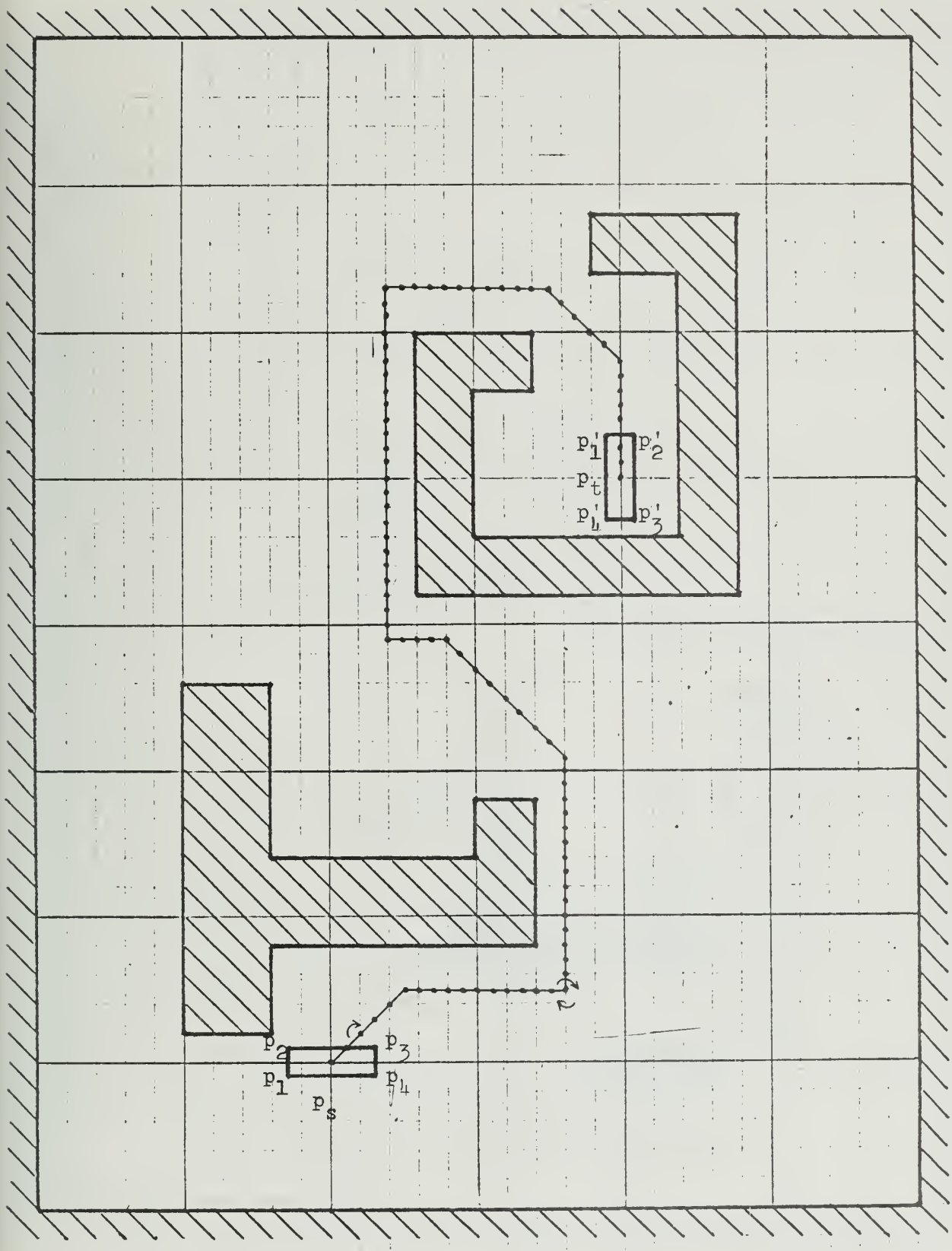


Fig. 34 $h_k = h_1$, $\omega = 0.2$, T_8 and R = {r;6}.

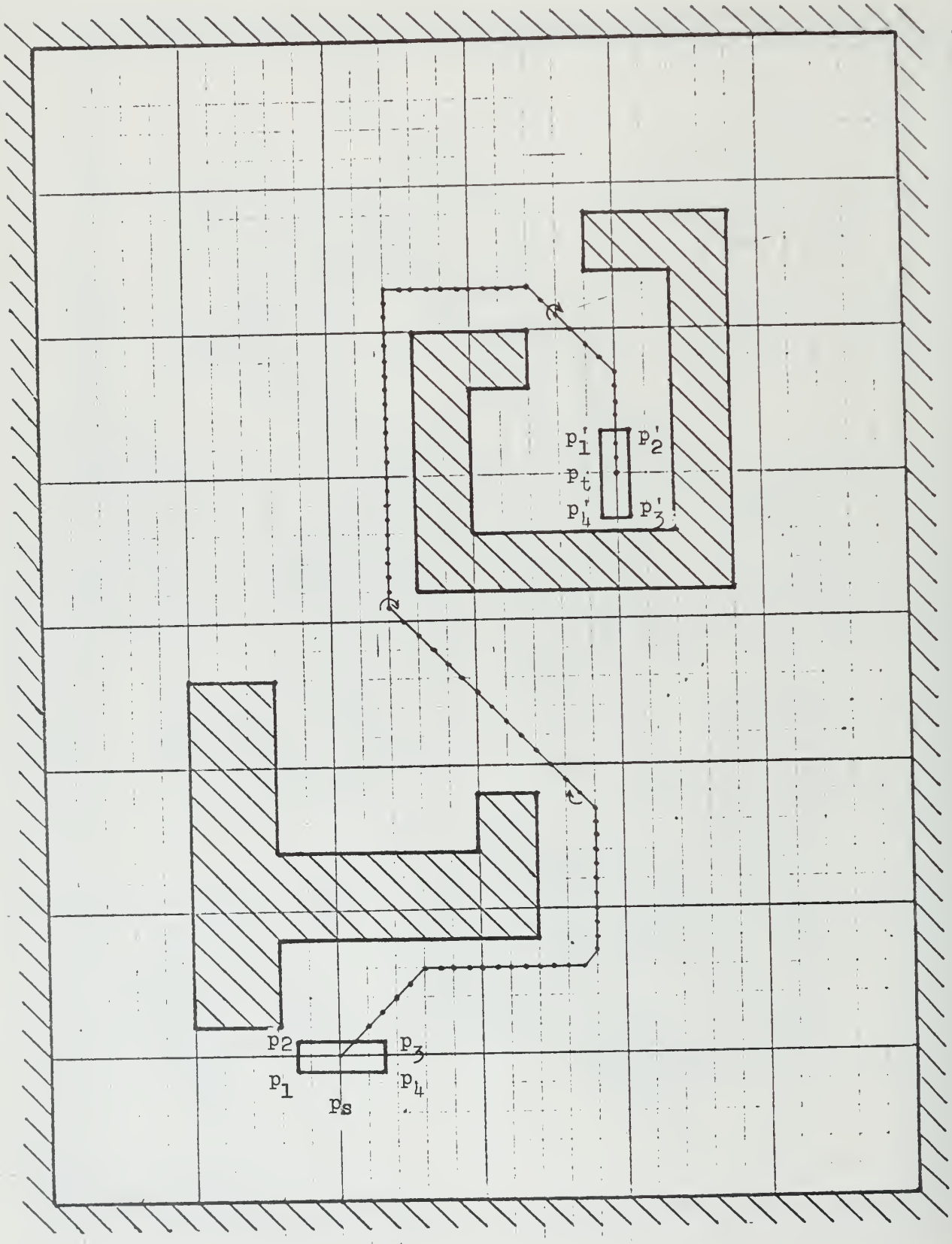


Fig. 35 $h_k = h_1$, $\omega = 0.8$, T_8 and R = {r;6}

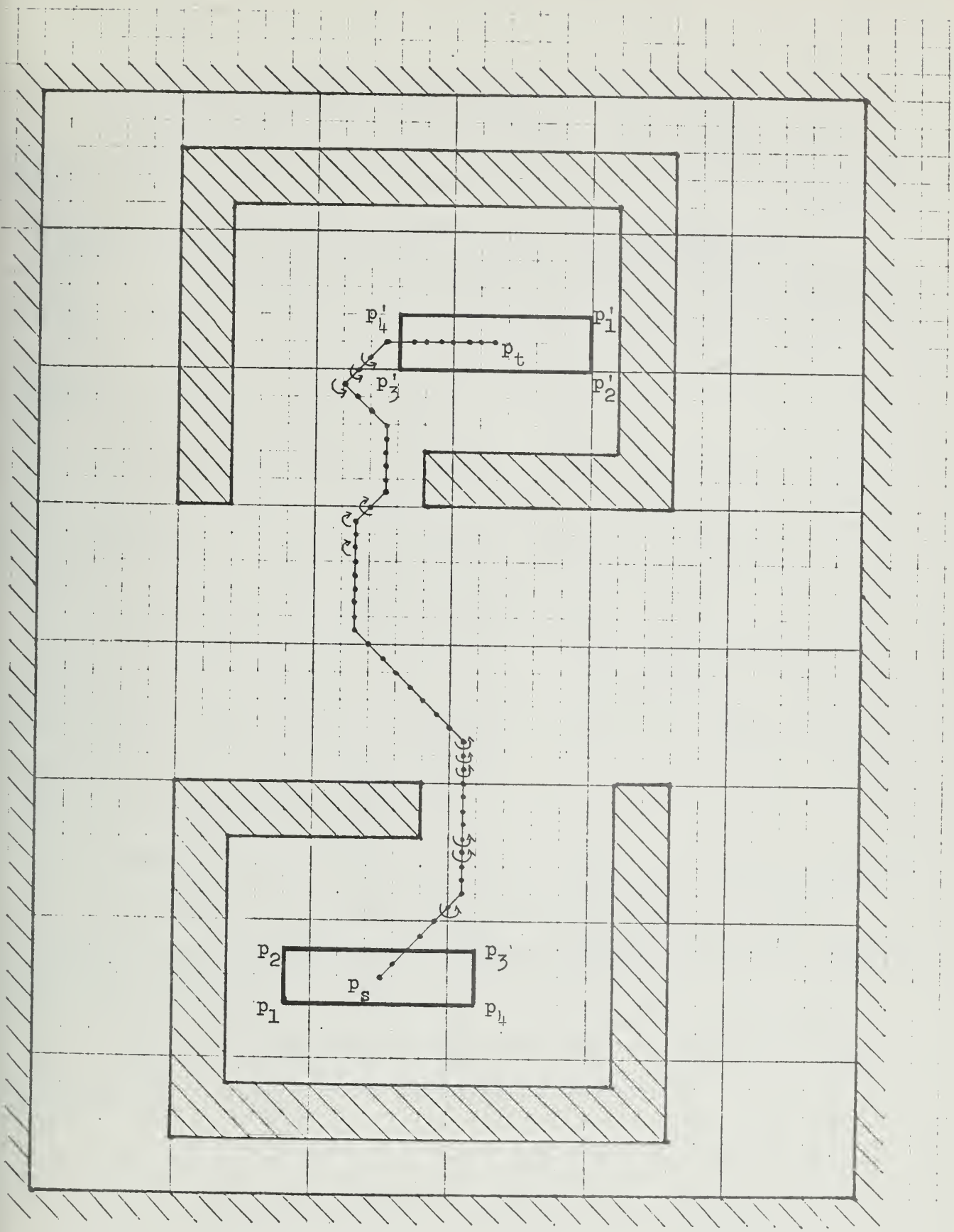


Fig. 36 $h_k = h_2$, $\omega = 0.5$, T_8 and $R = \{r;6\}$

The average $|\Gamma^*|/|\Gamma|$ is found to be greater than 85 percent and $|N|/|\Gamma^*|$ are located between 4 and 16, depending on the value of ω as well as the world C.

The program was written on PL/1 language and implemented on 360/75, the average run time of each step is about 30 to 80 miliseconds.

5. CONCLUSION

We have developed an intersection detection procedure for polyhedral objects by means of face-to-face intersection analysis. It first eliminates those faces of two objects which cannot possibly intersect each other. Such an elimination of faces, called "filtering", has been done by solution box approach and then by mutual divisibility between two faces as well as by mutual visibility if considered objects are convex. It then makes a pairwise analysis, such as parity mode detection, on the remaining faces.

From our experiment of running the Algorithm PD on various data, we conclude that the procedure developed in Chapter 3 is efficient not only for the determination of the placement of physical objects, but also for the simulation of moving objects in physically constrained space.

As an application of the intersection detection procedure developed, a modification of the Algorithm PD for polygonal objects was made and applied to solve the path finding problem in two-dimensional space. The method used for solving the problem may be modified relatively easily to solve the path finding problem in three-dimensional space.

Another approach for the solution of the path finding problem as well as for the solution of the original sofa problem [13, 24, 36] by means of "sequential pattern" representation of polygonal regions is our current interest and is being considered.

Each algorithm described in this report has been programmed in PL/1 and implemented on a 360/75 at the University of Illinois.

LIST OF REFERENCES

1. Comba, Paul G., "A Procedure for Detecting Intersections of Three-Dimensional Objects," Journal of the ACM, vol. 15, No. 3, July 1968, pp. 354-366.

2. Culbertson, J. T., "Some Uneconomical Robots," in Automata Studies, ed. by Shannon and McCarthy (Princeton Univ. Press, Princeton, N. J., 1956), pp. 99-115.

3. Doran, J. E., and Michie, D., "Experiments with the Graph Traverser Program," Proc. R. Soc. (A), 294, Sept. 1966, pp. 235-259.

4. Doran, J. E., "An Approach to Automatic Problem Solving," Machine Intelligence 1, Collins and D. Michie, editors, pp. 105-123.

5. Doran, J. E., "New Developments of the Graph Traverser," Machine Intelligence 2, ed. by E. Dale and D. Michie, Oliver and Boyd, Edinburgh, 1968, pp. 195-216.

6. Doran, J. E., "Experiments with a Pleasure Seeking Automaton," Machine Intelligence 3, ed. by D. Michie, Edinburgh University Press, Edinburgh, 1968, pp. 195-216.

7. Doran, J. E., "Planning by Generalization in a Simulated Robot," Machine Intelligence 4, ed. by D. Michie, Edinburgh University Press, Edinburgh, 19 , pp.

8. Eastman, Charles M., "Representation for Space Planning," Comm. of the ACM, vol. 13, No. 4, April 1970.

9. Ernst, H. A., "A Computer-Operated Mechanical Hand," Sc.D. Thesis, MIT, Department of Electrical Engineering, December 1961.

10. Floyd, R., "Algorithm 97, Shortest Path," Comm. of the ACM, No. 5, 1962, pp. 345.

11. Freeman, H., "On the Encoding of Arbitrary Geometric Configurations," IRE Trans. on Electronic Computers, vol. EC-10, June 1961, pp. 260-268.

12. Freeman, H., and Garder, L., "A Pictorial Jigsaw Puzzles: The Computer Solution of a Problem in Pattern Recognition," IEEE Trans. on Electronic Computers, vol. EC-13, April 1964, pp. 118-127.

13. Goldberg, M., "A Solution of Problem 66-11, Moving Furniture Through a Hallway," SIAM Review, vol. 11, No. 1, Jan. 1969, pp. 118-127.

14. Gorden, Armour, "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities," Man. Sci., Jan. 1963, pp. 244-309.

15. Hart, P., N. Nilsson, and B. Raphael, "A Formal Basis for Heuristic Determination of Minimum Cost Paths," Stanford Research Institute Report, (June 1967); and IEEE Trans. on Sys. Sci. and Cybernetics, (July 1968).

16. Homer, G. M., "Mobile Manipulator Systems," (in Proceedings of 14th Conference on Remote Systems Technology, 1966, American Nuclear Society).

17. Howden, W. E., "The Sofa Problem," Computer J. vol. 11, No. 3, Nov. 1968, pp. 299-301.

18. Kozdrowicki, E., "An Adaptive Tree Pruning System: A Language for Programming Heuristic Tree Searches," Proc. of the ACM, 1968, pp. 725-735.

19. Luh, Y. S. and R. J. Krolak, "Mathematical Model for Mechanical Part Description," Comm. of the ACM, vol. 8, No. 2, Feb. 1965.

20. Michie, D., "Strategy-Building with Graph Traverser," Machine Intelligence 1, 1962, pp. 135-152.

21. Michie, D., J. G. Fleming and J. V. Oldfield, "A Comparison of Heuristic, Iteractive and Unaided Methods of Solving a Shortest-Route Problem," Machine Intelligence 3, 1968, pp. 245-255.

22. Minsky, M., "Steps Toward Artificial Intelligence," Computers and Thought, E. Feigenbaum, and J. Feldman, Editors, (McGraw Hill Company, New York, New York, 1963), pp. 406-450.

23. Moore, E. F., "On the Shortest Path Through a Maze," Proceedings of the International Symposium on the Theory of Switching, Harvard Univ. Harvard Annals, vol. 3, 1959, pp. 285-292.

24. Moser, L., "The Problem 66-11, Moving Furniture Through a Hallway," is given SIAM Review, vol. 8, No. 3, July 1966, pp. 381.

25. Munson, J. H., "The SRI Intelligent Automaton Program," Artificial Intelligence Group, Technical Note 19, Jan. 1970.

26. Newell, A. and H. Simon, "GPS, a Program that Stimulates Human Thoughts," Computers and Thought, (McGraw Hill Company, New York, New York, 1963), pp. 279-293.

27. Newell, A., and G. Ernst, "The Search for Generality," Proceeding of IFIP Congress, (1965), pp. 17-22.

28. Nicholson, T., "Finding the Shortest Route Between Two Points in a Network," Computer Journal, 9, Nov. 1966, pp. 275-280.

29. Nilsson, N., "Preliminary Design of an Intelligent Robot," Computer and Information Science 2.

30. Pohl, Ira, "Bi-Directional and Heuristic Search in Path Problems," SLAC Report, No. 104, May 1969.

31. Pohl, Ira, "Heuristic Search Viewed as Path Finding in a Graph," Artificial Intelligence, vol. 1, No. 3, Fall 1970, pp. 193-204.

32. Raphael, B., "Programming A Robot," (Proceedings of International Federation for Information Processing Congress, 1968), pp. 1575-1581.

33. Rosen, Charles A., "An Experimental Mobile Automaton," Artifical Intelligence Group, Technical Note 39, SRI Project 8259, July 1970.

34. Samuel, A., "Some Studies in Machine Learning Using the Game of Checkers," Computer and Thought, (McGraw Hill Company, New York, New York, 1963), pp. 71-105.

35. Scoins, I., "Linear Graphs and Trees," Machine Intelligence 1, N. L. Collins and D. Michie (editors), 1967, pp. 3-15.

36. Sebastian, J., "A Solution of Problem 66-11, Moving Furniture Through a Hallway," SIAM Review, vol. 12, 1970, pp. 582- 586.

37. Slagle, J. R., and Philip Bursky, "Experiments with a Multipurpose Theorem Proving Heuristic Program," Journal of the ACM, vol. 15, No. 1, Jan. 1968, pp. 85-99.

38. Sutherland, I. E., "A Method for Solving Arbitrary-Wall Mazes by Computer," IEEE Trans. on Computers, vol. C-18, No. 12, Dec. 1969, pp. 1092-1097.

39. Sutro, L. L. and et. al., "Steps Toward the Automatic Command of Land, Sea, and Air Craft: Based on a Model of the Core of the Reticular Formation in Animals," R-635.

40. Talbot, J. E., "Programmable Industrial Robots," How they work, (in ISA Journal, Sept. 1966).

41. Warnock, John E., "The Hidden Line Problem and the Use of Halftone Displays," University of Utah, Computer Science, March 1969.

JUL 15 1971

JUL 15 1971

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.445-450(1971
Translator writing tools (TWT) prelimin
3 0112 088399677